FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání bakalářské práce

Název:	Aplikace na přípravu hudebních setů pro iOS a iPadOS
Student:	Lukáš Zima
Vedoucí:	Ing. Filip Glazar
Studijní program:	Informatika
Obor / specializace:	Softwarové inženýrství 2021
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	do konce letního semestru 2024/2025

Pokyny pro vypracování

Cílem práce je navrhnout a implementovat aplikaci pro DJs, která jim pomůže především se skládáním tracklistů, které poté slouží jako návod pro živé přehrávání skladeb. Aplikace bude napojena na některé nebo i více streamovacích služeb pomocí jejich API. Aplikace bude implementována pro operační systémy iOS a iPadOS. Pro implementaci použijte programovací jazyk Swift. Ideálně postupujte dle následujících kroků:

1) Analyzujte podobné aplikace
2) Navrhněte uživatelské rozhraní aplikace
3) Implementujte prototyp aplikace
4) Podrobte prototyp uživatelskému testování v rámci cílové skupiny
5) Na základě výsledků z testování navrhněte vhodné úpravy aplikace nebo je rovnou realizujte
6) Připravte aplikaci pro produkční nasazení

Elektronicky schválil/a Ing. Michal Valenta, Ph.D. dne 5. února 2024 v Praze.

Bakalářská práce

APLIKACE NA PŘÍPRAVU HUDEBNÍCH SETŮ PRO IOS A IPADOS

Lukáš Zima

Fakulta informačních technologií
Katedra softwarového inženýrství
Vedoucí: Ing. Filip Glazar
16. května 2024

Odkaz na tuto práci: Zima Lukáš. *Aplikace na přípravu hudebních setů pro iOS a iPadOS*. Bakalářská práce. České vysoké učení technické v Praze, Fakulta informačních technologií, 2024.

Obsah

Seznam obrázků

Seznam tabulek

Seznam výpisů kódu

Na prvním místě bych chtěl poděkovat především svému vedoucímu Ing. Filipu Glazarovi za vedení mé práce, cenné konzultace a poskytnutou zpětnou vazbu. Dále bych rád poděkoval své rodině, přítelkyni a přátelům, kteří mě při celém studiu i psaní bakalářské práce maximálně podporovali. Nakonec bych také rád poděkoval všem respondentům, kteří si našli čas poskytnout mi zpětnou vazbu.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací. Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů, zejména skutečnost, že České vysoké učení technické v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 citovaného zákona.

V Praze dne 16. května 2024

Abstrakt

Tato bakalářská práce se zabývá vývojem mobilní aplikace pro operační systémy iOS a iPadOS. Záměrem aplikace je umožnit diskžokejům lépe se připravit na své nadcházející živé hraní tím, že si hrané skladby mohou detailně naplánovat a rozvrhnout dopředu. V práci se postupně provádí analýza, návrh, implementace prototypu aplikace a uživatelské testování v rámci cílové skupiny. Aplikace je implementována v moderním frameworku SwiftUI za pomoci databázového frameworku SwiftData. Aplikace dále komunikuje se Spotify API a MusicAPI za účelem získání informací o skladbách. K provedení uživatelského testování byla využita platforma TestFlight. Výsledkem práce je užitečná a originální mobilní aplikace, která prošla testováním cílovou skupinou a která umožňuje diskžokejům lépe se připravit na svůj nadcházející DJ set.

Klíčová slova mobilní aplikace, iOS, uživatelské rozhraní, příprava hudebního setu, DJ, SwiftUI, Swift

Abstract

This bachelor thesis focuses on the development of a mobile application for the iOS and iPadOS operating systems. The intent of the app is to allow DJs to better prepare for their upcoming live shows by allowing them to plan and lay out the songs played ahead of time. The work consists of analysis, design, implementation of a prototype app, and user testing within the target audience. The application is implemented in the modern SwiftUI framework with the help of the SwiftData database framework. The application also interacts with Spotify API and MusicAPI to obtain song information. In order to perform user testing, the TestFlight platform was used. The result of the work is a useful and original mobile app which passed the target audience testing and which allows disc jockeys to better prepare for their upcoming DJ set.

Keywords mobile application, iOS, user interface, preparation of a music set, DJ, SwiftUI, Swift

Seznam zkratek

API	Application Programming Interface
BPM	Beats Per Minute
DAW	Digital Audio Workstation
DJ	Disk jockey
FR	Functional Requirement
FURPS	Functionality, Usability, Reliability, Performance, Supportability
ID	Identification
MVC	Model-View-Controller
MVVM	Model-View-ViewModel
PKCE	Proof Key for Code Exchange
URL	Uniform Resource Locator
USB	Universal Serial Bus

Úvod

S rozmáhající se digitalizací celého světa přirozeně došlo i na digitalizaci hudby. Už dávno není normou, že by se hudba musela hrát živě, aby ji někdo mohl slyšet. Dokonce už některé druhy hudby ani nemusí vytvářet celá kapela, stačí k tomu šikovný producent s přístupem k počítači. S tímto pokrokem se vyvinul i nový způsob živého přehrávání hudby – DJing. DJ – nebo diskžokej – je člověk, který pouští živě již vytvořené skladby a míchá je dohromady na vybavení k tomu určeném.

Já osobně se již od mládí zajímám o různé druhy elektronické taneční hudby a kromě samotného poslouchání hudby mě zaujal i DJing. Tento zájem se postupně rozvinul v koníček, kterému se věnuji již čtyři roky. Během této doby jsem nenarazil na mobilní aplikaci, která by umožňovala detailnější plánování DJského vystoupení – setu.

Tato práce se bude věnovat návrhu, implementaci a testování prototypu aplikace pro operační systémy iOS a iPadOS, která umožní DJům si naplánovat jejich živé vystoupení detailněji, než nabízí jiná momentální řešení. Uživatel si bude moci rozvrhnout set přesně tak, jak ho plánuje hrát živě, což mu umožní se na něj lépe připravit a ve výsledku mu samotné hraní zjednoduší. Testování aplikace bude provedeno v rámci cílové skupiny a z jeho výsledků budou navrženy budoucí úpravy a vylepšení aplikace.

Kapitola 1
Cíl práce

Cílem této práce je provedení analýzy, návrhu a implementace funkčního prototypu mobilní aplikace pro operační systémy iOS a iPadOS. Tato aplikace bude uživateli umožňovat několik dílčích funkcionalit:

Správa seznamů skladeb Uživatel bude moci vytvářet, upravovat a mazat seznamy skladeb.

Správa písniček Uživatel bude moci v daném tracklistu vkládat, upravovat a mazat písničky. Novou skladbu bude uživatel moci přidat následujícími způsoby:

- Dotazem na hudební API: Uživatel uvede jméno skladby a případně i jméno tvůrců a aplikace se pokusí takovouto skladbu nalézt dotazem na hudební API.
- Manuálním přidáním: Uživatel bude moci uvézt veškeré potřebné údaje o skladbě, čímž ji přidá.

U písniček bude uživatel moci upravovat jejich délku, začátek, konec a také bude moci měnit, kde se v tracklistu nachází.

Interaktivní úprava seznamu skladeb Po otevření či vytvoření nějakého seznamu skladeb bude uživatel přesměrován na interaktivní obrazovku, kde bude mít k dispozici 4 paralelní linky, mezi kterými bude schopen libovolně přemisťovat přidané písničky. Pro lepší orientaci a lehčí plánování bude také u každé linky zobrazena časomíra s minimální jednotkou 4 takty. Poloha skladeb se bude řídit touto časomírou.

Kromě samotné tvorby prototypu je dalším cílem podrobení aplikace uživatelskému testování v rámci cílové skupiny, kterou tvoří DJové. Na bázi výsledků tohoto testování budou navrženy vhodné úpravy aplikace.

Kapitola 2
Analýza

2.1 DJing a jeho kontext

Pojem DJ, anglicky disc jockey, byl vůbec poprvé použit v roce 1935. Tehdy sloužil jako označení pro rozhlasového moderátora. Během 50. let se DJing vyvinul do podoby, která je blízká té dnešní. V hlavním městě Jamaiky, Kingstonu, se začaly utvářet tzv. Sound Systémy. Jednalo se o uskupení DJů, kteří přinášeli do ulic hudbu, kterou nechtělo hrát radio. Hudbu přehrávali na gramofonech a pouštěli ji na velké reproduktory z kufrů dodávek. Mezitím se i v Evropě a Americe tato forma umění postupně popularizovala. [1, 2]

Od té doby se postupně z DJingu stal celosvětový fenomén. Analýza od Business Growth Reports odhaduje, že roku 2028 dosáhne trh s DJským vybavením hodnoty přes 19 miliard korun českých. [3] Report od společnosti IMS uvádí, že v roce 2023 činila DJská vystoupení 39% všech uskutečněných aktů na festivalech. Zároveň odhaduje, že taneční hudební průmysl dosahuje hodnoty přes 230 miliard korun českých. [4]

Pro lepší pochopení problematiky bych zde rád popsal druhy DJského vybavení:

2.1.1 Player

DJ player, nebo zkráceně player, je specializovaný přehrávač hudby, na kterém diskžokej načítá a pouští své skladby. U skladeb může upravovat jejich tempo, počátek přehrávání a další věci. Přehrávače lze historicky rozdělit na dva typy – vinylové a digitální. Vinylové přehrávače, nebo také gramofony, se k mixování hudby používaly výhradně, a to až do roku 2001. V tomto roce přišel Pioneer DJ s prvním CD přehrávačem svého druhu – CDJ-1000. [5] Od té doby se z vinylových přehrávačů většinově přešlo na ty digitální a CDJ přehrávače se staly jedničkou v tomto průmyslu. [6] Kromě pouštění hudby z CD disků se postupně u digitálních přehrávačů přidala podpora i pro SD karty či USB disky. Zároveň se postupně přidaly další pokročilejší funkcionality, jako je možnost periodického opakování části skladby (looping) nebo automatické srovnání dob u přehrávaných skladeb (Beat Sync).

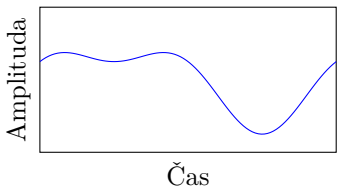
V rámci této práce se bere player jako přehrávač, který může v jednu chvíli přehrávat maximálně jednu skladbu. Toto tvrzení platilo všeobecně až do roku 2021, kdy Denon DJ, jeden z hlavních konkurentů Pioneeru, představil u svého nejnovějšího produktu funkci Dual Layer, která umožňuje hrát až dvě skladby současně na jednom zařízení. [7] Je to ale pořád dost ojedinělá funkce a vzhledem k tomu, že průmyslovým standardem jsou přehrávače od Pioneeru a ne přehrávače od Denonu, tak je vhodné se držet toho, že lze na každém playeru přehrávat maximálně jednu skladbu v jednu chvíli.

Obrázek 2.1 Nejnovější player od značky Pioneer DJ – CDJ 3000 [8]

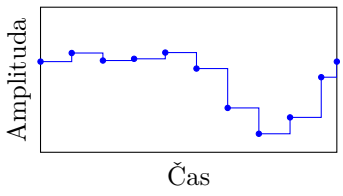
2.1.2 Mixer

Kromě možnosti načítat a pouštět skladby musí mít DJ i možnost u puštěných skladeb upravovat jejich hlasitosti a popřípadě provádět úpravy frekvenční charakteristiky či přidávat efekty. Zároveň se někde zvukové vstupy z playerů musí mísit, aby vytvořily ucelený mix. K těmto účelům existuje mixážní pult nebo mixer.

Tak jako playery, tak lze i mixážní pulty dělit dle způsobu, kterým zpracovávají zvuk, a to na analogové a digitální. Hlavním rozdílem mezi analogovým a digitálním signálem je, že se analogový signál přenáší v čase jako jedna spojitá vlna, zatímco digitální se přenáší v čase jako posloupnost funkčních hodnot. [9]



Obrázek 2.2 Příklad analogového signálu



Obrázek 2.3 Příklad digitálního signálu

Zároveň lze mixery dělit i podle jejich zaměření. Kromě mixerů pro DJe existují i mixery pro zvukaře či hudební producenty, ale ty nejsou pro tuto analýzu podstatné. Za první mixer určený pro DJe se dá považovat Rosie Mixer vynalezený zvukovým inženýrem Alexem Rosnerem pro newyorský klub Haven. Tento mixer umožňoval mísit až tři zvukové vstupy se schopností si kterýkoliv jednotlivý vstup pustit do sluchátek – cueing. Právě tato funkcionalita je jednou z klíčových u mixážních pultů určených pro DJe. [10]

V dnešní době nabízí DJ mixery zpravidla mnohem více. Jednou z klíčových vlastností je ekvalizér, jež umožňuje DJům upravit frekvenční spektrum hraných skladeb, například zvýšením basů nebo potlačením vysokých frekvencí, čímž dosáhne požadovaného zvukového profilu skladby. Další funkcí je crossfading, který umožňuje plynulé přechody mezi skladby hranými na různých kanálech mixeru tím, že postupně snižuje hlasitost jedné skladby, zatímco zvyšuje hlasitost druhé. Také často nabízí řadu zvukových efektů jako jsou ozvěna či frekvenční filtr.

Obrázek 2.4 Nejnovější 4-kanálový mixer od značky Pioneer DJ – DJM-A9 [11]

2.1.3 Controller

DJ controller je nejnovějším druhem DJského vybavení. Snaží se kombinovat mixer a playery do jednoho, kompaktního balení. Oproti klasické kombinaci mixer + playery představuje značně levnější způsob, jak hrát živě. Bývá také poměrně lehký a skladný, proto je ideální volbou pro začínající a cestující DJe.

Prvním komerčně úspěšným controllerem byl VCI-100 od firmy Vestax, který přišel na trh v roce 2007. Od té doby se tento druh DJského vybavení značně zpopularizoval. Controller se zpravidla zapojí do počítače, na kterém běží kompatibilní DJ software s knihovnou uživatelem pořízených skladeb, a umožní DJovi mixovat. Existují i dražší controllery, které nepotřebují propojit s počítačem a tudíž se v tomto ohledu chovají stejně jako kombinace mixeru s playery. [12]

Obrázek 2.5 Jeden z nejnovějších 4-kanálových controllerů od značky Pioneer DJ – DDJ-FLX10 [13]

2.1.4 Význam pro mou aplikaci

Pod pojmem naplánovat DJ set si lze představit různé postupy s rozdílnou mírou důrazu na detaily. Pro někoho může tento pojem znamenat pouhé vypsání skladeb v pořadí, v kterém je plánuje hrát. Nebere ohled na věci, jako jsou informace o čase nebo způsoby, kterými mixovat. Pro někoho jiného to zase může znamenat přivést každý jednotlivý přechod či mix mezi skladbami k dokonalosti s braním rytmiky a zvukových efektů v potaz, přesné naplánování časového rozložení skladeb a mnoho dalšího. A tuto skutečnost chce nějak zaznamenat.

Dalo by se tedy tvrdit, že se dá naplánovat jak obsluha playerů, tak obsluha mixeru. Má aplikace se bude zaměřovat především na to první. Uživatel bude moct naplánovat rozložení skladeb mezi playery v čase. Bude vědět, kdy a kde pustit novou skladbu nebo vypnout tu starou. Na naplánování aktivit spojených s obsluhou mixeru, jako je využití zvukových efektů či ekvalizéru, se má aplikace alespoň zatím zaměřovat nebude. DJ tedy bude vědět přesně, v který čas udělat mix, ale bude na něm, jakým způsobem ho vykoná.

2.1.5 Výpočty spojené s teorií hudby

Délka skladeb je na internetu zpravidla udávaná v časových jednotkách, jako jsou minuty a sekundy, což je pro většinu případů vhodné. Hudební teorie, která se za těmito skladbami skrývá, využívá jiný systém. V tomto systému figurují jako hlavní jednotky času takty a doby. [14]

Takt je základním stavebním kamenem každé hudební skladby. Jedná se o krátký časový úsek, který je dále členěn na doby – stejně dlouhé časové úseky. Existují takty dvoudobé, třídobé atd. V elektronické taneční hudbě se v drtivé většině případů pracuje s taktem čtyřdobým. [14, 15]

Jako most mezi světem hudební teorie a světem minut a sekund slouží tempo, nebo přesněji BPM (Beats Per Minute). Tempo určuje rychlost skladby. Zpravidla ji určuje pomocí BPM, což je počet dob za minutu. S těmito znalostmi lze sestavic vzorec:

$$\text{Počet taktů} = \frac{\text{čas}}{k} \times \frac{\text{BPM}}{\text{taktové označení}}$$

kde:

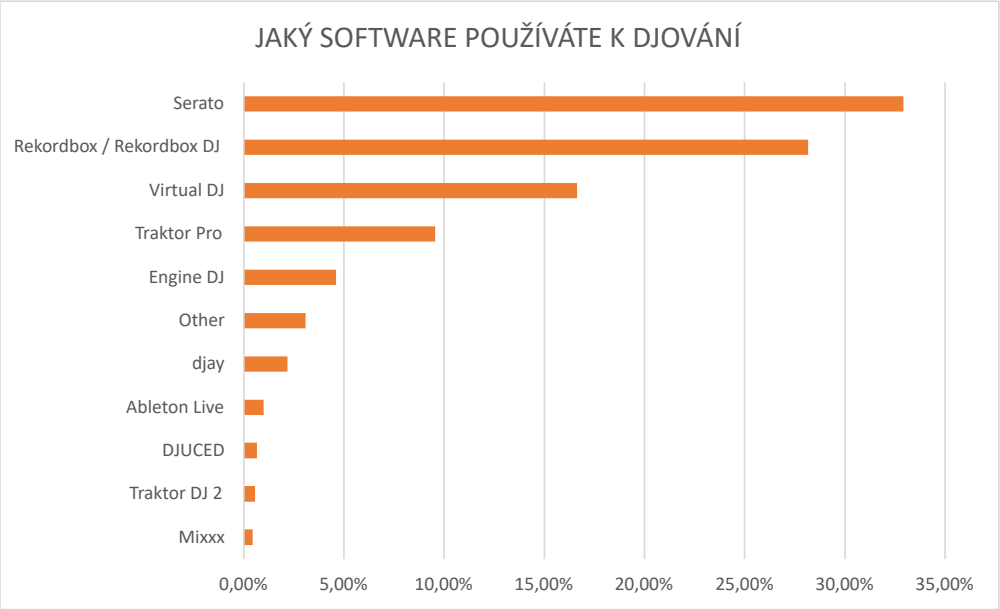
- čas je množství času v nějaké jednotce (ms, s, m),
- BPM je tempo v dobách za minutu,
- taktové označení značí počet úderů/dob v taktu, zpravidla 4,
- k je koeficient, který závisí na jednotce času:
  - $k = 60000$ pro milisekundy,
  - $k = 60$ pro sekundy,
  - $k = 1$ pro minuty. [14]

Díky tomuto vzorci bude má aplikace moct přeskakovat mezi oběma časovými systémy a využívat tak oba. To je žádoucí, jelikož elektronické skladby mají pravidelnou strukturu a DJové se při mixování řídí právě takty a frázemi. Fráze jsou tvořeny čtyřmi nebo osmi takty a jsou to dílčí úseky skladby, mezi kterými se zpravidla něco může (ale nemusí) stát – Pokles či nárůst energie, přidání nové melodie či instrumentu… Naopak uvnitř jedné fráze se zpravidla žádná drastická změna neděje a elementy v ní zní předvídatelně a periodicky. [15]

2.2 Analýza existujících aplikací

Během své čtyřleté praxe jsem se nesetkal s aplikací, která by byla svou funkčností či designem srovnatelná s touto. Já osobně jsem k nějaké pokročilejší přípravě používal Notes, do kterých jsem si zapisoval informace o průběhu plánovaného setu. Takovéto řešení je ale velice zdlouhavé a nenabízí žádnou přidanou hodnotu pro DJe. Je to jednoduše poznámkový blog.

I přesto se ale několik aplikací, především takových, které podporují tvorbu nějakého seznamu písniček – playlistu či tracklistu, dá považovat za podobné. Ještě konkrétněji se lze zaměřit na aplikace, které slouží DJům k přípravě a exportu písniček pro živé hraní. Hlavními z nich jsou dle průzkumu 2.6 Serato a Rekordbox.



Obrázek 2.6 Dotazník bezmála 20 tisíc DJů na jejich oblíbený software, 2023 [16]

Dalším typem aplikací, v kterých lze podobné funkcionality dosáhnout, je DAW. DAW, neboli Digital Audio Workstation, je softwarová platforma určená k nahrávání, úpravě, mixování a produkci hudby. Tyto systémy nabízí širokou škálu nástrojů k úpravě a produkci zvuku, jako jsou virtuální syntetizéry, nástroje a efekty. Dá se říct, že je to takové hudební studio, co běží na vašem počítači. [17]

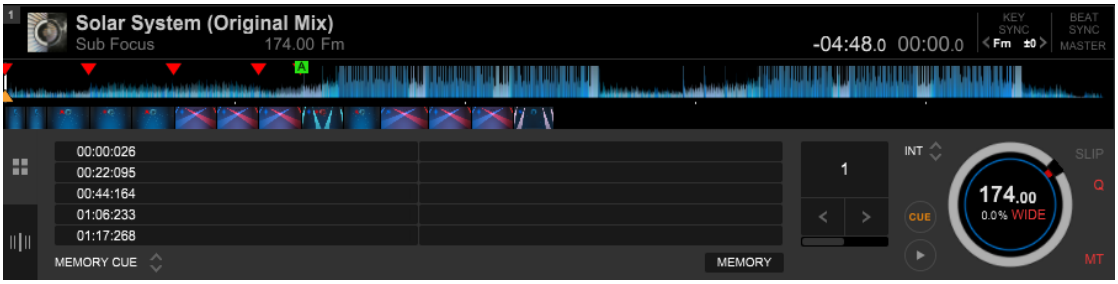
Úplnou novinkou je aplikace DJ.Studio, která vyšla v roce 2023. Kategorizují se jako DAW pro DJe a nabízí kompletní řešení pro tvorbu mixů a DJ setu jako celku. Aplikaci zpracovávané v této práci se podobá nejvíce. [18]

2.2.1 Serato DJ a Rekordbox

Serato DJ a Rekordbox jsou dva navzájem si konkurující DJ softwary vyvíjené společnostmi Serato a AlphaTheta.[1] Přestože AlphaTheta oznámila roku 2023 akvizici společnosti Serato, tak jsou oba softwary i nadále vyvíjeny nezávisle na sobě. [20] Oba softwary mají kromě počítačové verze i mobilní verzi, ale ta umožňuje víceméně jenom živě přehrávat a mixovat skladby přehrávané z počítače, což není něco, na co se soustředí mnou vyvíjená aplikace. Počítačová verze nabízí DJům kompletní řešení pro mixování hudby, analýzu skladeb, správu hudební knihovny a další funkce, které usnadňují živé vystoupení a tvorbu setů. Skladby se kromě manuálního vložení dají přidat i nalezením přes služby jako jsou Beatport či Soundcloud. Serato a Rekordbox mají sice své rozdíly, ale pro účely této práce se dají považovat za skoro totožné.

V rámci funkcí, které tyto softwary nabízí, je tou klíčovou pro kontext této práce nastavení takzvaných cue bodů na jednotlivých skladbách. Těmito body si typicky DJ označí důležité části skladby, jako je její začátek, build-up a drop, ale může si tyto body vytyčovat i v pravidelných úsecích, například každých 16 taktů. V tom, jestli body použije a popřípadě kde je vytvoří, má uživatel úplnou volnost. Mezi těmito body je pak možno libovolně přeskakovat. Dají se představit jako takové záložky v knize, které vám umožní okamžitě se dostat k vybranému místu. Pomáhají DJům se lépe orientovat ve skladbě, která právě hraje, ale také jim umožňují pustit skladbu jinde, než na jejím úplném začátku. [21]

[1] Dříve Pioneer DJ. [19]



Obrázek 2.7 Cue body vytyčené každých 16 taktů před prvním dropem, s jedním cue bodem navíc, který je 8 taktů před dropem, Rekordbox

Další klíčovou funkcí je možnost sestavení tracklistů. Tyto tracklisty si pak může DJ exportovat na svůj USB disk, který potom jen stačí zapojit do kteréhokoliv DJského vybavení, které tuto funkcionalitu podporuje. V opačném případě si musí nosit svůj notebook, který si s vybavením propojí a již zmíněný software mu umožní hrát a mixovat naživo.

	#	Preview	Artwo	Track Title	Key	Artist
CUE	1			Brana remix v2 4	F#m	
CUE	2			It's That Time (Dimension Remix)	Fm	Marlon Hoffsta
CUE	3			Wilkinson-Frontline	Fm	
CUE	4			I Want It (Original Mix)	Em	Bennie & Nu El
CUE	5			Gold Dust (Tsuki Remix)	Dm	
CUE	6			Love To Me (Original Mix)	Ab	Dimension
CUE	7		WILKINSON	Take You Higher (Original Mix)	Fm	Wilkinson
CUE	8			Techno (Original Mix)	Fm	Dimension
CUE	9			Whip Slap (Original Mix)	Fmaj	Dimension

Obrázek 2.8 Krátký tracklist vytvořený v programu Rekordbox

Tyto dvě funkce zkombinované dohromady by teoreticky umožňovaly rozumně si naplánovat set. Díky sestavenému tracklistu ví DJ v jakém pořadí skladby pouštět a díky vytyčeným cue bodům by věděl, kdy skladby zahrát. Takovéto využití cue bodů je ovšem nepraktické, jelikož by si je DJ musel přemisťovat pokaždé, když by chtěl skladbu zahrát od jiného bodu, než byl ten předchozí. Proto je standardní si cue body vytyčovat spíše v důležitých částech skladby či pravidelných úsecích, aby toto rozestavení bodů DJ nemusel nikdy v budoucnu měnit. To ovšem způsobí, že konkrétní body, v kterých v nějakém setu spustit nějakou skladbu, si musí pamatovat.

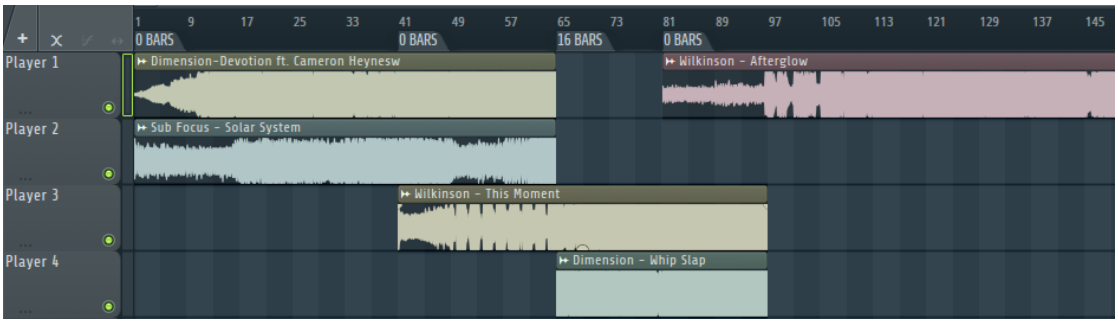
2.2.2 Software DAW

DAWs jsou využívány k produkci, nahrávání a úpravě hudby po celém světě. Využívají je jak amatérští producenti, tak špičkoví profesionálové, kteří je mohou využívat v kombinaci s fyzickými nástroji. Mezi nejpoužívanější DAWs patří Logic Pro, Ableton a FL Studio. [16] Rozdíly mezi konkrétními DAWs jsou poměrně velké, ale pro účely porovnání s funkcionalitami mé aplikace jsou zanedbatelné.

Jedním ze společných rysů těchto aplikací je možnost tvorby neomezeného počtu zvukových stop, na které lze pokládat zvukové nahrávky v čase. Právě díky této funkcionalitě by se dal DJ set relativně spolehlivě naplánovat podobným způsobem, jakým to nabízí mnou vyvíjená aplikace. DJ by mohl veškeré skladby, které plánuje zahrát, vložit do některého z těchto softwarů a celý set si na zvukové stopy vyskládat. Například pokud by plánoval hrát na čtyřech přehrávačích, použil by čtyři zvukové stopy. Skladby by ořízl tak, aby zbyla jenom ta část, kterou chce hrát. Některé ze zmíněných aplikací mají i mobilní verze, takže by tento proces mohl provézt i na mobilu.

Takovéto řešení by ale pořád neumožňovalo si jednoduše vizuálně vyznačit, v kterých bodech pustit kterou skladbu. DAWs umožňují pokládání značek s popisy do zvukových stop, ale celý

proces by se tím prodloužil. Zároveň by si DJ musel takto vyskládané stopy nějak vyfotit či uložit, aby se na ně mohl při živém hraní dívat. Pokud by tento proces chtěl podstoupit na mobilní verzi DAW, musel by veškeré skladby mít stáhnuté na mobilním telefonu, což by představovalo další zábranu. Klasické DAWs jsou navrženy s odlišnými cíli než je plánování DJ setu, a proto nenabízí tolik vymožeností, které by tento proces usnadnily.



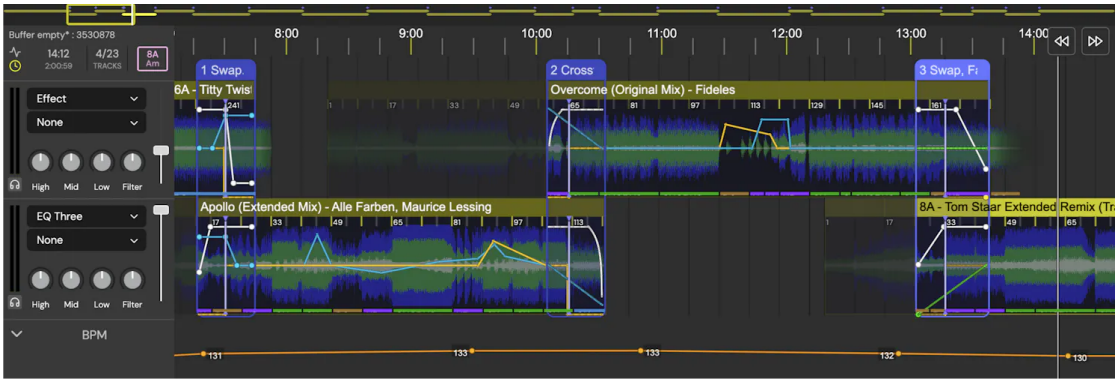
Obrázek 2.9 Krátký tracklist vytvořený v DAW, FL Studio

2.2.3 DJ.Studio

DJ.Studio je počítačová aplikace vydaná teprve před rokem, tedy v roce 2023. Klade si za cíl umožnit DJům nejen perfektně připravit set, ale také ho zahrát. Pokud DJ nechce plánovat set manuálně, aplikace by ho měla naplánovat za něj s pomocí umělé inteligence. Připravený set pak umožňuje rovnou zahrát na živo či nahrát, přičemž DJ nepohne ani prstem. [18]

Funkcionalita aplikace DJ.Studio, která je pro tuto analýzu zásadní, je možnost přípravy tracklistu. DJ.Studio podporuje nalezení skladby z několika zdrojů, jako jsou Youtube, Beatport, Rekordbox, Serato a další. Podporuje také manuální vložení skladby z počítače. Skladbu se poté pokusí zanalyzovat, aby určila tempo, délku, tóninu, fráze… [18, 22]

Skladby se pokládají na časovou osu podobnou té, kterou lze nalézt v kterémkoliv DAW. U skladeb lze upřesňovat, jak budou mezi nimi zahrané přechody a v jakých částech skladby se bude co dít. Aplikace sice podporuje dva playery, ale má ještě dvě další místa pro hraní krátkých zvukových stop, která se dají použít jako další dva playery. [18, 23]



Obrázek 2.10 Krátký 2-playerový mix vytvořený v aplikaci DJ.Studio [18]

DJ.Studio je díky výše zmíněným vlastnostem užitečným a komplexním nástrojem pro DJe. Nemá ovšem mobilní aplikaci a je placené, konkrétně 199 Eur za zjednodušenou verzi a 499 Eur za kompletní verzi. Tato cenovka je pravděpodobně z velké části daná schopností aplikace opravdu

zahrát daný mix a připravit set pomocí umělé inteligence. Obě z těchto funkcí ale nejsou účelem mnou vyvíjené aplikace.

2.2.4 Shrnutí

Z výše provedené analýzy vychází, že momentálně neexistuje aplikace, která by splňovala každý z funkčních požadavků mé aplikace. Nejvíce se tomu blíží DJ.Studio, které ovšem nemá mobilní aplikaci. V tabulce níže jsou shrnuty vlastnosti zkoumaných aplikací. U každé aplikace je napsané, jak moc danou vlastnost splňuje:

- Ano plně splňuje.
- Částečně splňuje.
- Ne nesplňuje.

Tabulka 2.1 Porovnání analyzovaných aplikací

Vlastnost	Serato a Rekordbox	DAW	DJ.Studio
Manuální vložení skladeb	Ano	Ano	Ano
Vyhledání skladeb přes online služby	Ano	Ne	Ano
Tvorba seznamu skladeb	Ano	Částečně	Ano
Rozdělení skladeb mezi playery	Ne	Ano	Ano
Nastavení bodů, v kterých skladby zahrát	Částečně	Částečně	Ano
Mobilní aplikace	Ne	Částečně	Ne

2.3 Požadavky

K určení a identifikaci požadavků vycházím převážně z tabulky 2.1. Tyto požadavky a požadavky s nimi související v této sekci popíšu a rozřadím. Nejprve je ale nutné uvézt, co to požadavky v kontextu vývoje vůbec jsou.

Požadavky jsou v oblasti vývoje softwaru seznam schopností, funkcionalit a podmínek, které musí systém splňovat. Hlavními výzvami při sepisování požadavků jsou jejich správná a přesná identifikace a zaznamenání tak, aby byly lehce pochopitelné jak vývojářem, tak klientem. [24]

Jedním z široce užívaných modelů, dle kterého lze kategorizovat a blíže specifikovat požadavky, je model FURPS+. [24]

Functionality Funkce a vlastnosti systému.

Usability Použitelnost – Lidský faktor, dokumentace.

Reliability Spolehlivost systému.

Performance Výkon – Rychlost, doba odezvy, přesnost.

Supportability Podporovatelnost či rozšiřitelnost.

Každý požadavek bude mít svůj daný formát. U požadavků budu zaznamenávat následující informace:

Název Konkrétní a výstižný název požadavku.

Popis Dostatečně detailní popis chování a schopností, kterých se tento požadavek týká.

Priorita Určuje důležitost tohoto požadavku v kontextu celé aplikace. Priority budu dělit do následujících kategorií:

- Esenciální – Aplikace požadavek musí splnit.
- Vysoká – Aplikace by požadavek opravdu měla splnit.
- Střední – Aplikace by požadavek měla splnit.
- Nízká – Aplikace by požadavek mohla splnit, ale ne nutně v této době.

Typ Způsob kategorizace požadavku, v rámci této práce se zaměřím na dva typy:

Funkční požadavek Týká se chování aplikace v nějaké dané situaci – její funkcionality a dovednosti.

Nefunkční požadavek Vše ostatní – Kvalitativní požadavky, dokumentace, architektura…

2.3.1 Funkční požadavky

FR1 – Přehled všech vytvořených tracklistů

Uživatel bude prezentován s obrazovkou, na které bude seznam tracklistů, které dříve vytvořil. Tracklist je seznam skladeb, které DJ plánuje během svého setu zahrát. Skladby jsou v něm zpravidla seřazeny v takovém pořadí, v jakém je plánuje zahrát naživo. U každého tracklistu bude viditelný jeho název a datum založení. Na této obrazovce bude také možnost vytvořit nový tracklist. Po rozkliknutí či vytvoření nějakého tracklistu ho aplikace přesměruje na obrazovku úpravy tracklistu.

Priorita: Esenciální

FR2 – Vytvoření nového tracklistu

Uživatel bude mít možnost vytvořit nový tracklist stisknutím odpovídajícího tlačítka, přičemž ho aplikace přesměruje na obrazovku, v které bude moci uvézt požadované informace o tracklistu jako je jeho plánovaná délka nebo tempo hrané hudby. Po uvedení všech potřebných informací ho aplikace převede na obrazovku tvorby tracklistu.

Priorita: Esenciální

FR3 – Smazání tracklistu

Uživatel bude moci na obrazovce přehledu všech tracklistů nějaký tracklist vhodným gestem smazat.

Priorita: Vysoká

FR4 – Podpora až čtyř playerů

Aplikace bude umožňovat rozplánovat si tracklist napříč až čtyřmi přehrávači, což uživateli umožní předem určit, na kterém přehrávači a kdy se má přehrát konkrétní skladbu. Tím se mu zjednoduší hraní na živo. Rozdělení do čtyř přehrávačů bude začleněno do obrazovky pro úpravu tracklistu.

Volba čtyř přehrávačů jako horní hranice v aplikaci vychází z běžných nastavení DJských sestav, které obvykle obsahují dva až čtyři přehrávače. Možnost použít více než čtyři přehrávače je v praxi vzácná a většina DJů takovou kapacitu nepotřebuje, což činí čtyři přehrávače ideálním standardem pro tuto aplikaci. Nicméně, plány na budoucí rozšíření aplikace mohou zahrnovat podporu pro větší počet přehrávačů, aby se vyhovělo potřebám a požadavkům všech uživatelů.

Priorita: Esenciální – Právě tato funkcionalita odděluje tuto aplikaci od aplikací pro tvorbu obyčejných seznamů písniček.

FR5 – Vložení nové skladby do tracklistu

Uživatel bude mít možnost vložit novou skladbu do momentálně upravovaného tracklistu dvěma způsoby:

1. Uživatel zadá jméno skladby a nepovinně i jména umělců, kteří ji vytvořili. Aplikace se pokusí takovou skladbu nalézt dotazem na hudební API a nalezené skladby zobrazí. Uživatel si bude moci některou z těchto skladeb vybrat a po zkontrolování či doplnění povinných údajů se nově vytvořená skladba vloží do tracklistu.
2. Uživatel uvede všechny potřebné informace o skladbě a aplikace takovouto skladbu vytvoří a vloží do tracklistu.

Skladba se v obou případech vloží na konec daného tracklistu.

Priorita: Esenciální

FR6 – Vymazání skladby z tracklistu

Uživatel bude moci při upravování tracklistu libovolnou skladbu vhodným gestem smazat.

Priorita: Esenciální

FR7 – Změna délky skladby

Uživatel bude moci při upravování tracklistu libovolnou skladbu vhodným gestem zkrátit nebo prodloužit. Bude moci změnit od jakého bodu skladbu plánuje pustit a v jakém bodě ji plánuje vypnout.

Priorita: Vysoká

FR8 – Libovolné přesouvání skladeb v rámci jednoho tracklistu

Aplikace bude při upravování tracklistu umožňovat libovolně přemisťovat skladby. To znamená, že uživatel bude moci měnit jejich pořadí a čas zahrání libovolně, a to i napříč více DJ playery. Tato funkcionalita bude umožněna vhodným gestem, preferovaně přes Drag & Drop.

Drag & Drop je druh operace či gesta v rámci uživatelského rozhraní, která umožňuje uživateli uchopit nějaký objekt, přetáhnout jej na požadované místo a pustit ho tam. Puštěný objekt pak s požadovaným místem provede nějakou interakci, v případě této aplikace se na něj přemístí. [25]

Priorita: Esenciální

FR9 – Časová osa tracklistu

Uživatel bude mít možnost zjistit aktuální informace o svém plánovaném setu, jako je jeho délka nebo časy, v kterých má pustit či vypnout jednotlivé písničky. Tyto informace bude moci vyčíst z časové osy, která bude začleněna v obrazovce úpravy tracklistu.

Minimální jednotkou této časové osy budou 4 takty. Tato volba je založena na tom, že právě čtyři nebo osm taktů tvoří ve většině skladeb frázi. Pro přehled o čase vyjádřeném v minutách bude časová osa též obsahovat ukazatele každých 10 minut.

Priorita: Vysoká

FR10 – Automatické přichytávání skladeb k časové ose

Skladba by se po kterékoliv operaci s ní – tedy po přidání, přesunutí, zkrácení či prodloužení – měla na obrazovce úpravy tracklistu zarovnat k nejbližšímu bodu na časové ose. Tento bod odpovídá nejbližšímu ukazateli čtyř taktů. Tímto zarovnáváním se ulehčí DJům otázka frázování, jelikož aplikace rovnou předpokládá, že skladby budou pouštěny vždy od začátku nějaké fráze a vypínány na konci nějaké fráze.

Priorita: Střední

FR11 – Export tracklistu do textového formátu
Aplikace umožní uživateli exportovat vytvořený tracklist do textové reprezentace, v které bude vidět posloupnost písniček včetně informací o tom, jak mají být hrány dohromady.
Priorita: Nízká

FR12 – Nastavení
Aplikace bude mít sekci nastavení, v kterém si uživatel bude moci změnit různé aspekty aplikace jako jsou barevné kombinace, minimální jednotka časové osy, font a další.
Priorita: Nízká

2.3.2 Nefunkční požadavky

NR1 – Podpora iOS a iPadOS verze 17 a novější
Aplikace bude podporovaná – plně funkční – na operačních systémech iOS a iPadOS verze 17 a novější.
Priorita: Esenciální

NR2 – Zabezpečení API klíčů
Aplikace bude pracovat s API klíčy k přístupu k vybraným hudebním API. Proto je nutné zajistit, že bude tyto klíče i bezpečně ukládat, aby se k nim útočník nemohl lehce dostat.
Priorita: Střední – Aplikace nebude sbírat žádné údaje o uživatelích, takže k úniku citlivých dat nemůže nikdy dojít.

NR3 – Podpora různých velikostí obrazovek
Aplikace bude své uživatelské rozhraní dynamicky škálovat pro každou velikost obrazovky u iPhonů a iPadů a při tom se zachová veškerá funkcionalita.
Priorita: Střední

NR4 – Podpora anglického jazyka
Uživatelské rozhraní aplikace bude vedeno v anglickém jazyce, jelikož se angličtina dá považovat za univerzální jazyk. Podpora jiných jazyků, jako je ten český, může být také implementována v rámci FR12.
Priorita: Střední

Kapitola 3
Návrh

3.1 Technologie

V této sekci budou shrnuty a porovnány technologie, které jsem k vývoji mobilní aplikace pro operační systémy iOS a iPadOS mohl použít. Vzhledem k omezenosti možností, které Apple vývojářům dává, je výběr značně limitovaný. U každé technologie popíšu, čím se zabývá a jak funguje, a odůvodním, proč jsem danou technologii zvolil či nikoliv.

3.1.1 iOS

iOS je operační systém, který spravuje a vyvíjí společnost Apple exkluzivně pro své mobilní telefony – iPhony. Tento operační systém poprvé představila v roce 2007 spolu s představením prvního iPhonu. Od svého uvedení se iPhone a s ním spojený iOS staly nevídaně úspěšnými. Odhaduje se, že v roce 2023 iPhone vlastnilo kolem 18% populace. [26, 27]

iOS se vyznačuje svou jednoduchostí ovládání, rychlým a spolehlivým výkonem, špičkovým zabezpečením a proslulým designem, který se vyskytuje v obdobné formě i napříč ostatními operačními systémy od společnosti Apple, jako jsou macOS, ipadOS, watchOS a další. Apple totiž usiluje o co nejužší propojení všech jejich operačních systémů za účelem co nejpříjemnějšího uživatelského zážitku. Tato realita vede k vytvoření tzv. Apple ekosystému, uvnitř kterého mnoho uživatelů zůstane už navždy. [28]

Do roku 2019 běžel iOS i na tabletech společnosti Apple – iPadech. Roku 2019 byl představen nový operační systém přímo pro iPady – iPadOS. Svým vzhledem silně připomíná iOS a vyvíjet lze díky úzké provázanosti na oba operační systémy zároveň bez větších problémů. Nasazení své aplikace tedy cílím na iOS i iPadOS. [26, 29]

3.1.2 Programovací jazyky

Vyvíjet aplikaci pro iOS nativně lze primárně ve dvou programovacích jazycích, kterými jsou Swift a Objective-C. Existují i jiná řešení umožňující vývoj napříč operačními systémy (cross-platform), jako jsou Flutter nebo Kotlin Multiplatform Mobile, ale nejsou podporovaná společností Apple a jsou často mladá a nedokončená. Proto jsem tyto možnosti nevyužil. [30]

3.1.2.1 Objective-C

Jazyk Objective-C vznikl v osmdesátých letech minulého století. Jakožto primární jazyk pro vývoj aplikací pro operační systémy značky Apple se stal poté, co roku 1996 Apple odkoupil

společnost NeXT, kterou vedl Steve Jobs. NeXT vyvíjela svůj software v tomto jazyce a Apple v tom pokračoval. [31]

Tento jazyk byl navržen jako jednoduchá, ale užitečná nadstavba nad jazykem C. Umožňuje objektově orientované funkcionality po vzoru Smalltalku, jednoho z prvních objektově orientovaných jazyků. Zároveň přináší dynamic runtime, který umožňuje měnit chování tříd za běhu programu. Jednou z vlastností dynamic runtime je, že volání metod jsou dynamicky vázána. Skutečná implementace, která je volána při poslání zprávy objektu, je určena v okamžiku volání na základě dynamického typu tohoto objektu. [32, 33]

V dnešní době se Objective-C využívá spíše jen k údržbě již existujících starších aplikací. Nemá moderní rysy a nenabízí zdaleka tolik vymožeností oproti jeho nástupci Swift. Proto jsem tento jazyk nezvolil k vývoji mé aplikace.

3.1.2.2 Swift

Swift byl představen společností Apple roku 2014 jako přímý nástupce jazyku Objective-C. Od té doby začal postupně nahrazovat Objective-C pro vývoj aplikací na zařízení od Apple a v dnešní době se dá tvrdit, že jej plně nahradil. [34]

Swift je moderní programovací jazyk, který se zaměřuje na srozumitelnou syntaxi, bezpečný kód a výkonnou kompilaci. Snaží se vyjít vývojářům maximálně vstříc ve tvorbě bezpečného kódu bez ztráty na rychlosti běhu kódu. [35] Toho dosahuje využitím mnoha moderních přístupů, jako jsou:

Automatická správa paměti Swift se o správnou a včasnou destrukci alokované paměti stará sám a vývojáře tím nezatěžuje. K tomu využívá deterministické počítání referencí a prostředek uvolní, jakmile není využíván. [35]

Typová bezpečnost a inference typů Swift klade velký důraz na typovou bezpečnost, což vývojářům pomáhá předcházet chybám spojeným s nesprávným použitím datových typů. Díky pokročilému systému inference typů je Swift schopen odvodit datový typ proměnné z kontextu, což umožňuje vývojářům psát kód, který je zároveň bezpečný a čistý, bez potřeby explicitně deklarovat typ každé proměnné. [35]

Bezpečná manipulace s hodnotou nil Objekty standardně nemohou nabývat hodnoty nil, což eliminuje mnoho běžných chyb spojených s neinicializovanými proměnnými nebo neexistujícími referencemi. V případech, kdy je nil jako možná hodnota žádoucí nebo potřebná, zavádí Swift Optionals, jež slouží jako volitelné objekty. Umožňují vývojářům explicitně deklarovat tuto možnost a vyžadují od nich, aby bezpečně zpracovali i potenciálně neexistující hodnoty. [35]

Protocol a Extension Swift umožňuje vývojářům definovat Protocol, který specifikuje metody a vlastnosti, jež musí třída či struktura z něj vycházející implementovat. Extensions pak umožňují přidávat novou funkcionalitu k existujícím typům bez potřeby dědit z těchto typů nebo je upravovat. Společně tyto dvě funkčnosti podporují flexibilní design a znovupoužitelnost kódu. [36]

Funkcionální prvky Swift zahrnuje prvky funkcionálního programování, jako jsou funkce vyššího řádu map, filter a reduce, které umožňují vývojářům efektivně manipulovat s kolekcemi a provádět složitější transformace dat s minimálním množstvím kódu. [37]

Z výše zmíněných důvodů byl Swift jasnou volbou pro vývoj mé aplikace.

3.1.3 Frameworky uživatelského rozhraní

Každá mobilní aplikace musí mít vhodnou vizuální podobu, a proto je kromě programovacího jazyka důležité zvolit také nástroj, s kterým lze efektivně vytvářet uživatelská rozhraní. Apple nabízí dva frameworky, které slouží jako nástroje k tvorbě uživatelského rozhraní – UIKit a SwiftUI.

3.1.3.1 UIKit

UIKit byl zveřejněn společností Apple v roce 2008, krátce po odhalení prvního iPhonu. Byl vytvořen za účelem zjednodušení vývoje na iOS tím, že nabízí již připravené komponenty pro zpracování uživatelských vstupů, dotyků a dalších interakcí. Zároveň nabízí širokou škálu grafických prvků jako jsou tlačítka, seznamy, tabulky…[38]

Tento framework je dodnes aktivně vyvíjen a vylepšován společností Apple a jeho hlavní výhoda lpí v jeho stáří. Využívá ho rozsáhlá komunita vývojářů a je k němu sepsaná bohatá dokumentace. Má imperativní syntaxi, tudíž se vývojář musí postarat sám o jakékoliv události, změny a aktualizace uživatelského rozhraní. Díky tomu má vývojář skoro neomezené možnosti a dá se tvrdit, že v UIKitu lze vytvořit v podstatě cokoliv. [38]

Právě imperativní syntaxe a stáří tohoto frameworku byly důvody, proč jsem si ho k vývoji mé aplikace nevybral. Vývojář zodpovídá za veškeré dění v aplikaci, což vede k velkému množství napsaného kódu a výrazně zpomaluje vývoj. Současně se skoro nekonečnými možnostmi přichází i významná náročnost procesu učení se vyvíjet v tomto frameworku. V těchto ohledech byl SwiftUI lepší volbou.

3.1.3.2 SwiftUI

SwiftUI je mladý framework představený společností Apple v roce 2019. Má sloužit jako alternativa k UIKitu, která se s ním ovšem dá v mnoha případech kombinovat. Hlavním rozdílem od UIKitu je jeho deklarativní syntaxe. Vývojář přímo definuje, jak má uživatelské rozhraní vypadat a co má dělat. O vnitřní fungování, jako jsou změny a aktualizace stavů, se pak za něj stará SwiftUI. Proces vývoje je díky tomu intuitivní a rychlý. [39]

Vzhledem ke svému mládí tento framework nenabízí tolik možností jako UIKit. Apple na něm usilovně pracuje a každým rokem přidává nové funkcionality, ale přesto je ještě nedokončený a vývojář nemá úplnou volnost v tom, co může vytvořit. Z tohoto důvodu ale existuje možnost kombinace se starším UIKitem, který tuto volnost poskytuje. Přechod ze SwiftUI do UIKitu není jednoduchý, ale v případech potřeby lze implementovat.

SwiftUI jsem si vybral jako framework pro mou aplikaci. Deklarativní syntaxe a moderní povaha jazyku umožňují rychlejší vývoj, než v UIKitu. Současně je patrné, že Apple má v plánu tento framework v budoucnosti dále zdokonalovat tak, aby mohl výhledově nahradit UIKit.

```
import SwiftUI

struct AlbumDetail: View {
    var album: Album

    var body: some View {
        List(album.songs) { song in
            HStack {
                Image(album.cover)
                VStack(alignment: .leading) {
                    Text(song.title)
                    Text(song.artist.name)
                        .foregroundStyle(.secondary)
                }
            }
        }
    }
}
```

Obrázek 3.1 Příklad deklarativní syntaxe ve SwiftUI [39]

3.1.4 Persistence dat

Aplikace bude muset mít schopnost ukládat různá data o skladbách, tracklistech atd. V iOS existuje několik možností, jak toho dosáhnout. Kromě využití databázových služeb je možné data zapisovat do souborů, ukládat je v cloudu, nebo je odesílat přes vlastní backendové API.

Ukládání dat do souborů je neefektivní, neboť vyžaduje manuální správu a formátování souborů. Tento přístup také nenabízí žádné vestavěné mechanismy pro optimalizaci výkonu, jako je lazy loading nebo indexace, což může vést k pomalejším načítacím časům a zvýšenému využití paměti.

Využití cloudu či napsání backendového API by bylo užitečným řešením, které by kromě persistence dat nabídlo i synchronizaci dat mezi zařízeními. Nevýhodami těchto služeb jsou závislost na internetovém připojení a náročnější implementace. Takovéto řešení by bylo žádoucí, pokud by má aplikace potřebovala data sdílet mezi zařízeními, což není aktuálním cílem.

Jako ideální kompromis se jeví nativní databázové technologie pro iOS. Tyto technologie umožňují efektivní integraci databázových operací s kódem. Zároveň nabízí schopnost synchronizace napříč zařízeními díky službě CloudKit. Existují dvě – Core Data a SwiftData. [40, 41]

3.1.4.1 Core Data

Core Data je databázový framework využívající SQLite databázi, jež běží na každém iOS zařízení. Byl vydán společností Apple v roce 2005. Jádrem Core Data je možnost abstrakce a manipulace s daty na úrovni objektů namísto přímé práce s databázovými tabulkami. To výrazně usnadňuje vývojářům práci s daty tím, že umožňuje definovat datové modely v grafickém editoru a automaticky generovat kód pro práci s těmito modely. [40, 42]

Core Data také zajišťuje správu paměti objektů, jejich vztahů a persistenci, což zahrnuje ukládání, načítání, aktualizace a mazání dat. Umožňuje pokročilé databázové operace a synchronizaci dat mezi uživateli iCloudu přes CloudKit. [40]

Výše zmíněné operace jsou umožněny souborem následujících tříd, které tvoří dohromady tzv. Core Data Stack:

NSManagedObjectModel který reprezentuje datový model aplikace, definující její typy, atributy a vzájemné vztahy.

NSManagedObjectContext který sleduje změny instancí spravovaných typů.

NSPersistentStoreCoordinator který ukládá a načítá instance typů z datového uložiště.

NSPersistentContainer který inicializuje všechny tři zmíněné třídy najednou. [43]

Tento framework má prudkou křivku učení a ovládnout ho tak, aby fungoval bezproblémově, může být komplikované. Zároveň existuje novější řešení, které si klade za cíl bezproblémovou integraci se SwiftUI. Proto jsem Core Data nezvolil.

3.1.4.2 SwiftData

SwiftData, podobně jako SwiftUI, je nový framework, který si klade za cíl vylepšit či nahradit ten stávající. Apple ho představil teprve před rokem na Worldwide Developers Conference 2023. [44]

Podobně jako SwiftUI, tak i SwiftData nabízí moderní deklarativní přístup k psaní kódu. Díky integraci s jazykem Swift umožňuje vývojářům dotazovat a filtrovat data přímo pomocí kódu, což zajišťuje plynulou integraci s projekty využívajícími SwiftUI. Modelování entit je intuitivní a efektivní, neboť využívá běžné Swift typy, které se označí deklarací @Model, což eliminuje potřebu spravovat či modelovat v jiných nástrojích. [41]

SwiftData se snaží automaticky odvozovat vztahy mezi entitami, podle kterých si vygeneruje schéma. Jakékoliv změny těchto entit za běhu aplikace pak automaticky propisuje do databáze a zároveň notifikuje SwiftUI o změně těchto dat. [41]

Díky novotě tohoto frameworku je v mnoha ohledech nedodělaný. Nenabízí zdaleka tak komplexní možnosti ani dokumentaci jako Core Data. Pro komplikovanější kontrolu nad daty může vývojář využít ModelContext, což je třída zodpovědná za veškeré operace s entitami. V tomto ohledu lze nalézt podobnost s Core Data, což není překvapující, jelikož SwiftData framework vznikl jako nadstavba nad Core Data. [41]

Vybral jsem tuto technologii, protože ulehčuje a urychluje věci okolo práce s databází a spolu se SwiftUI nabízí četné výhody. Pro účely mé aplikace nečiní nedokončenost tohoto frameworku zásadní překážku.

```
import SwiftData
@Model
class Recipe {
    @Attribute(.unique) var name: String
    var summary: String?
    var ingredients: [Ingredient]
}
```

Obrázek 3.2 Příklad modelování entity ve SwiftData [41]

3.1.5 Uživatelské testování

3.1.5.1 TestFlight

TestFlight je součástí služby App Store Connect, která nabízí vývojářům kompletní řešení pro vydávání, správu a testování svých aplikací pro software společnosti Apple. TestFlight konkrétně umožňuje provádět interní i externí uživatelské testování prototypů aplikací. Interní testování probíhá mezi členy vývojového týmu a externí probíhá mezi kýmkoliv, koho vývojář pozve. [45, 46]

Pozvání externího testera je jednoduché, stačí mu poslat odkaz. Pokud má tester nainstalovanou aplikaci TestFlight, otevření odkazu ho automaticky přidá jako externího testera a prototyp mu nainstaluje. Testeři mají možnost podání zpětné vazby a to jak v podobě slovní, tak i v podobě fotek obrazovky. [46]

K TestFlight existují alternativy jako je Firebase od společnosti Google, ale ty nenabízí zdaleka tolik možností a jednoduchosti jako nativní TestFlight. Firebase konkrétně ke svému fungování potřebuje, aby tester uvedl své Unikátní identifikační číslo zařízení a vývojář ho do systému zadal, což je nepříjemnost navíc pro testera. Apple má přísná opatření ohledně instalace jimi neověřených aplikací a proto bylo žádoucí jít cestou nejmenšího odporu, i když užívání TestFlight vyžaduje Apple Developer účet, který vývojáře stojí 2 799 KČ ročně. [47, 48]

3.2 Architektura

Softwarová architektura je soubor architektonických či návrhových prvků, které mají určitou danou podobu. Rozlišujeme tři různé třídy architektonických prvků:

- Zpracovávající prvky,
- datové prvky,
- spojovací prvky. [49]

Zpracovávající prvky jsou komponenty, které zajišťují transformaci datových prvků. Datové prvky obsahují informace, které software používá a které se transformují. Spojovací prvky (které někdy mohou být buď prvky zpracování, nebo datové prvky, nebo obojí) jsou pojivem, které drží různé části architektury pohromadě. [49]

V této sekci budou shrnuty a porovnány dva architektonické návrhové vzory, s kterými se v aplikacích vyvíjených ve SwiftUI lze setkat. Jedná se o Clean Architecture a MVVM (Model-View-ViewModel). Je důležité podotknout, že se zde nenachází jinak populární MVC (Model-View-Controller), jelikož deklarativní a reaktivní podstata frameworku SwiftUI tomuto návrhovému vzoru odporuje.

3.2.1 Clean Architecture

Robert C. Martin, dlouholetý softwarový inženýr a autor mnoha knih na toto téma, tvrdí, že rozdělením softwaru do vrstev a dodržováním pravidla závislosti vytvoříte systém, který je ze své podstaty testovatelný, se všemi výhodami, které z toho vyplývají. Pravidlo závislosti říká, že pouze vnější vrstvy smí obsahovat závislosti na ty vnitřní, nikdy naopak. O tom pojednává Clean Architecture. [50]

Tento přístup tedy navrhuje rozdělit software do několika dílčích komponent, kde má každá svůj pevně daný úkol. V případě mobilní aplikace by se vrstvy daly rozdělit na:

- Prezentační – tuto roli zastávají typy, které zobrazují uživateli data.
- Podniková – tuto roli zastávají typy, které se starají o veškeré logické úkony v aplikaci.
- Datová – tuto roli zastávají typy, které se starají o přístup k datům, například z databáze nebo z internetu.

Vývoj aplikace s touto architekturou je časově náročnější, jelikož vyžaduje mnoho abstrakce a kódu navíc z důvodů budoucí rozšiřitelnosti a dodržení pravidla závislosti. Přináší ovšem řadu užitečných konceptů, kterými se při implementaci můžu inspirovat.

3.2.2 MVVM

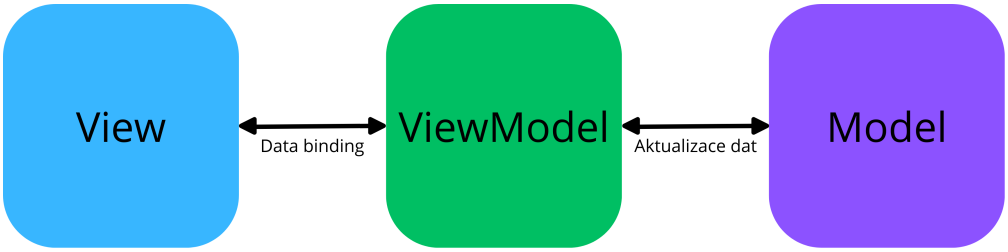
MVVM (Model-View-ViewModel) poprvé představil v roce 2005 softwarový architekt společnosti Microsoft John Gossman. Jedná se o variaci MVC pro moderní deklarativní uživatelská rozhraní. Za vývoj těchto rozhraní by nemusel být zodpovědný vývojář, ale například návrhář. [51, 52]

MVVM, tak jako Clean Architecture, pomáhá oddělit podnikovou a datovou vrstvu od té prezentační. MVVM definuje tři komponenty:

- Model – typ, který zapouzdřuje data aplikace. Patří sem definice dat i přístup k nim.
- View – typ, který zobrazuje uživateli data a reaguje na jeho akce.
- ViewModel – typ, který funguje jako prostředník mezi Modelem a View. ViewModel obsahuje stavové informace a logiku z View, na které může View reagovat, ale neobsahuje žádnou logiku pro přímou manipulaci s uživatelským rozhraním. Zároveň se stará o manipulaci s Model typy. [52]

Klíčovým prvkem, který umožňuje MVVM fungovat, je data binding, což je mechanismus, který propojuje data mezi View a ViewModelem. Díky tomu může View reagovat na změny ve ViewModelu automaticky bez nutnosti manuálně aktualizovat uživatelské rozhraní. Jedná se o jednu z funkcionalit deklarativních frameworků jako je SwiftUI. [52]

Rozhodl jsem se použít MVVM hned z několika důvodů. Prvním z nich je rychlost a přímočarost vývoje. Zpravidla má každá obrazovka svůj View a každý View má svůj ViewModel. Operace s daty pak lze řešit různými způsoby, ale tento základ zůstává. Současně je MVVM široce používanou architekturou pro SwiftUI aplikace, takže existuje mnoho učebních a informačních materiálů. Pro mou aplikaci nabízí ideální kompromis mezi efektivním vývojem a důkladným rozdělením aplikace na dílčí nezávislé komponenty.



Obrázek 3.3 Ilustrace MVVM [53]

Kapitola 4
Implementace

V implementační kapitole této práce občas používám ve výpisech kódu symbol „...“ k vynechání některých méně důležitých či opakujících se částí kódu. Tento způsob jsem zvolil ke zvýšení čitelnosti kódu a lepšímu zaměření se na důležité části.

4.1 Databáze

K databázovým operacím jsem dle podsekce 3.1.4 využil SwiftData. Tento framework vyžadoval krátké přednastavení v podobě konfigurace ModelContaineru a jemu příslušející ModelConfiguration.

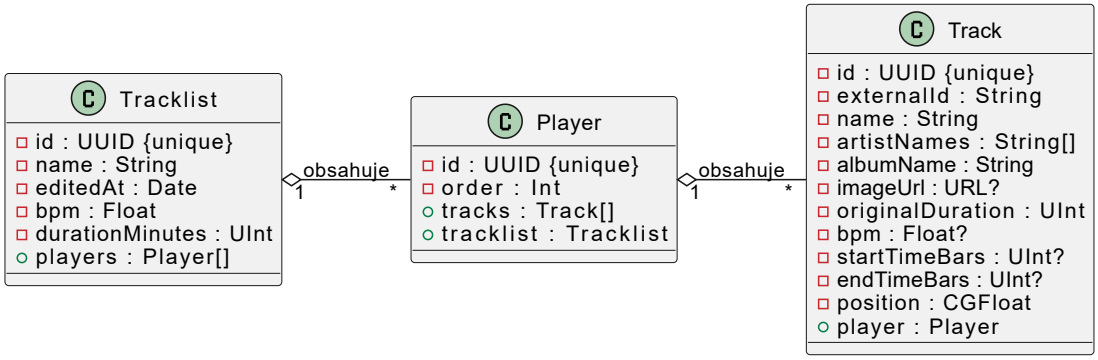
```
let tracklist = Tracklist.mockTracklist1()
let sharedModelContainer: ModelContainer = {
    let schema = Schema([
        Tracklist.self,
    ])
    let modelConfiguration = ModelConfiguration(schema: schema,
                                                isStoredInMemoryOnly: true)
    do {
        let container = try ModelContainer(for:schema,
                                           configurations: [modelConfiguration])
        container.mainContext.insert(tracklist)
        let player1 = Player.mockPlayer1()
        let player2 = Player.mockPlayer2()
        ...
        container.mainContext.insert(player1)
        container.mainContext.insert(player2)
        ...
        player1.tracklist = tracklist
        player2.tracklist = tracklist
        ...
        tracklist.players!.forEach { player in
            player.tracks!.forEach { track in
                track.player = player
            }
        }
        return container
    } catch {
        fatalError("Error initializing")
    }
```

Výpis kódu 4.1 Konfigurace ModelContaineru s mock daty

V tomto kroku jsem zároveň do kontextu databáze mohl vložit zkušební (mock) data, která mi ulehčila návrh a vývoj uživatelského rozhraní. Další položkou, která mi ulehčila vývoj, byla možnost vytvoření databáze jenom dočasně v paměti stroje. Tato položka je v kódu nazývána isStoredInMemoryOnly. Vzniklý sharedModelContainer se se všemi položkami předá aplikaci při každém spuštění.

4.1.1 Doménový model

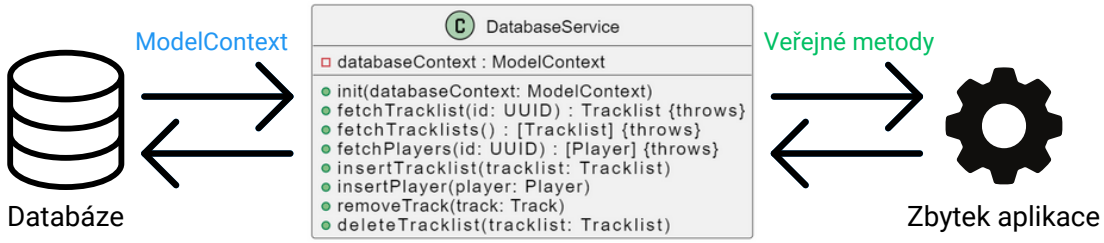
Ke správnému fungování aplikace je nutno definovat entity a vztahy mezi nimi. V mém případě mi stačilo definovat tři entity – Tracklist, Player a Track. U každé entity jsem dále identifikoval, které údaje budu potřebovat ukládat.



Obrázek 4.1 Diagram tříd, které tvoří doménový model mé aplikace

4.1.2 DatabaseService

Z důvodů sjednocení přístupu k databázi a lepšího rozdělení zodpovědností jsem vytvořil třídu DatabaseService, která využívá ModelContext pro získávání, přidávání a mazání dat. Tato myšlenka byla inspirovaná návrhovým vzorem Repository.



Obrázek 4.2 Ilustrace DatabaseService

4.2 Hudební API

V této sekci budu popisovat a rozebírat hudební API, která jsem v aplikaci použil včetně implementačních detailů. Využití hudebního API umožňuje uživateli jednoduše vyhledat skladbu, kterou chce přidat. Data o skladbě, jako její celý název, délku, fotku a tempo potom aplikace získá sama ze zmíněného API. Je to jedna z nejdůležitějších funkcionalit, které musí má aplikace obsahovat.

Snažil jsem se vybrat taková hudební API, která by dohromady nabízela co nejširší a nejrozmanitější katalog skladeb bez většího vzájemného překrývání. Konkrétně jsem API při výběru rozdělil do dvou kategorií:

API nabízející oficiálně vydanou hudbu Do této kategorie patří API známých streamovacích služeb jako jsou Spotify, Apple Music, Youtube Music, Deezer… Zároveň sem patří služby, které umožňují nákup skladeb jako Beatport či Juno Download. Hudba je na těchto platformách komercializována a často orientována na masový trh, což způsobuje, že se katalogy jednotlivých služeb ve velké míře překrývají. Proto jsem se rozhodl z této kategorie vybrat jen jedno API, a tím je Spotify Web API.

Ostatní hudební API Do této kategorie bych zařadil API takových služeb, které umožňují komukoliv vydat hudbu, ne nutně oficiálně. Umělci na těchto platformách jsou často nezávislí. Konkrétně mám na mysli služby Soundcloud a Bandcamp.

Soundcloud je široce používanou službou mezi DJi, jelikož se na ní nachází mnoho kvalitní hudby, kterou nelze zpeněžit a tudíž vydat oficiálně. Jedná se například o bootlegy, což jsou neautorizované remixy skladeb, kterých zpeněžení by porušovalo autorské zákony. Současně je nahrání skladby na SoundCloud přístupné každému, a tudíž se zde nachází hudba od začínajících umělců, kteří ještě nevydávají komerční cestou. Tato služba představuje pro DJe ideální naleziště nedoceněné hudby.

Díky široké využívanosti mezi cílovou skupinou mé aplikace a relativně nízké míře překryvu s ostatními službami by byl Soundcloud ideální volbou pro druhé API do mé aplikace. Soundcloud nabízí pro vývojáře Soundcloud API, které ovšem momentálně nepřijímá nové registrace aplikací. Tuto skutečnost jsem mohl částečně obejít užitím MusicAPI, které přístup k Soundcloud API nabízí. [54, 55]

4.2.1 Spotify Web API

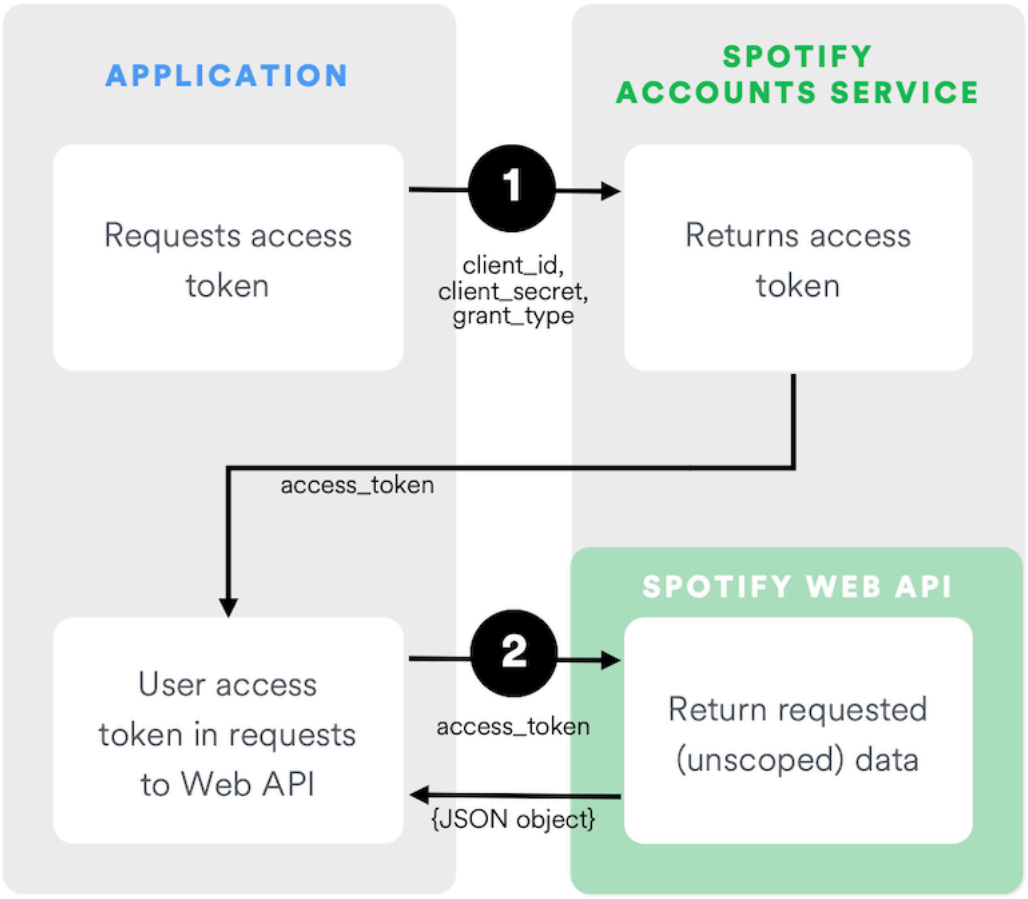
Spotify Web API poskytuje vývojářům bohatou sadu nástrojů, které umožňují integrovat Spotify do jejich aplikací. Toto API nabízí bezplatný přístup k široké škále dat Spotify, včetně informací o skladbách, albech a umělcích. Vývojáři mohou prostřednictvím autentizačního procesu pracovat i s účtem a daty konkrétního uživatele, což otevírá možnosti pro správu playlistů, analýzu hudebních preferencí a další. Autentizace uživatele ovšem pro mou aplikaci není nutná, jelikož má aplikace potřebuje pouze vyhledávat skladby v databázi Spotify. [56]

Proces registrace aplikace k využívání tohoto API byl jednoduchý. Stačilo se přihlásit svým Spotify účtem a založit novou aplikaci, přičemž jsem ihned obdržel potřebné přihlašovací údaje (Client ID a Client secret). Tyto údaje slouží k autorizaci aplikace, aby mohla přistupovat k datům od Spotify.

Spotify k autorizaci využívá rámec OAuth 2.0, který definuje čtyři typy udělení oprávnění/autorizace (flows):

- Authorization code,
- Authorization code s PKCE rozšířením,
- Client credentials,
- Implicit grant. [57]

Já jsem pro svou aplikaci zvolil Client credentials. Funguje tak, že si aplikace zažádá o přístupový token zasláním svých přihlašovacích údajů na koncový bod (endpoint) /api/token. S pomocí tohoto tokenu se pak dotazuje na hudební API. Tento typ autorizace funguje jednoduše a nevyžaduje přihlášení od uživatele, což ostatní 3 typy vyžadují. [57, 58]



Obrázek 4.3 Client credentials flow [58]

Má aplikace využívá tři endpointy tohoto API – /search, /tracks/id a /audio-features/id. /search umožňuje vývojáři vyhledávat hudbu v katalogu Spotify podle klíčových slov nebo frází, které zadá do parametru q. Tento koncový bod podporuje vyhledávání v různých kategoriích, jako jsou skladby, umělci, alba a playlisty. Pro specifikaci kategorie může vývojář využít parametr type. [59]

```
curl --request GET \
  --url 'https://api.spotify.com/v1/search?q=Reload&type=track' \
  --header 'Authorization: Bearer 1POdFZRZbvb...qqillRxMr2z'
```

Výpis kódu 4.2 Příklad provedení dotazu na skladbu Reload na /search endpoint

Spotify API na tento dotaz ve výchozím nastavení vrátí až dvacet výsledků. V tomto případě by byl každý výsledek jedna skladba. Limit, kterým omezit počet výsledků, se nastavuje parametrem limit. [59]

K vyhledání konkrétnějších informací o skladbě slouží endpoint /tracks/id, kde id značí unikátní identifikátor skladby pro Spotify API. Tento endpoint vrátí všeobecné informace o skladbě jako jsou její název, umělci, album, fotka a popularita. [60]

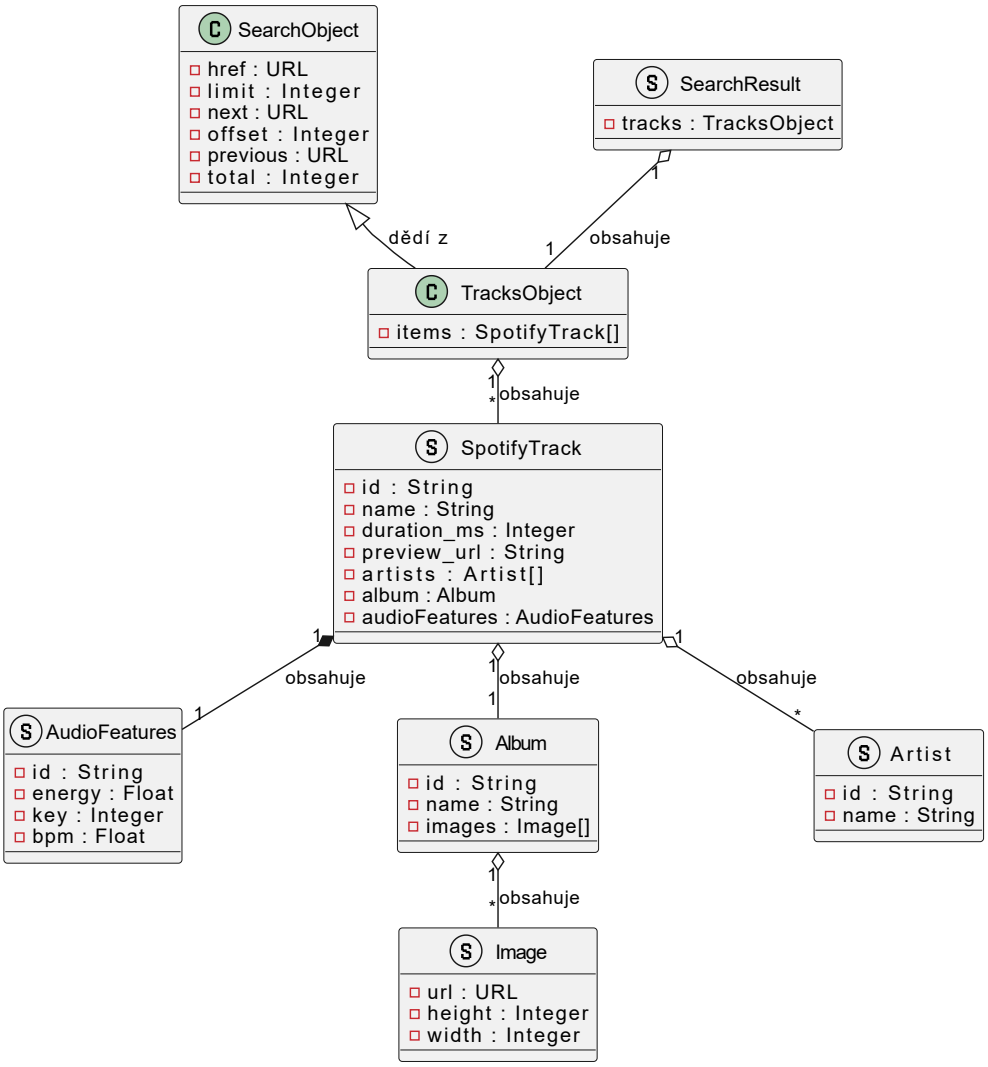
Endpoint /audio-features/id umožňuje získat detailnější informace o zvukových vlastnostech skladby identifikované svým id. Tyto informace jsou částečně získány vlastní analýzou skladby od Spotify a jejich správnost není zaručena. Mezi tyto informace patří například tempo, tónina,

energie, hlasitost a instrumentálnost skladby. Pro mou aplikaci je nejvíce relevantní možnost získání tempa (bpm) skladby. [61]

```
curl --request GET \
  --url https://api.spotify.com/v1/audio-features/25zILlagMmi16cFCsL5QtR \
  --header 'Authorization: Bearer 1POdFZRZbvb...qqillRxMr2z'
```

Výpis kódu 4.3 Příklad provedení dotazu na skladbu Reload na /audio-features/id endpoint. Reload má id 25zILlagMmi16cFCsL5QtR

Formát vrácených výsledků je vždy jasně definovaný v dokumentaci ke Spotify Web API. Kompletní diagram tříd, které jsem k reprezentaci skladby a audio-features k ní náležejících využil, je na obrázku 4.4:



Obrázek 4.4 Diagram tříd ke Spotify Web API

4.2.2 MusicAPI

MusicAPI je nově vyvinuté API od týmu společnosti FreeYourMusic, která již léta pomáhá uživatelům přenášet playlisty mezi streamovacími službami. Toto API bylo vytvořeno za účelem usnadnění integrace široké škály hudebních streamovacích služeb do jakékoli aplikace nebo platformy. S MusicAPI mohou vývojáři centralizovaně přistupovat ke službám jako jsou Spotify, Apple Music, YouTube Music, Tidal, Soundcloud a dalších devatenáct podporovaných služeb. [62]

Toto API nabízí bezplatnou variantu pro aplikace mající méně než 500 uživatelů. Proces registrace a získání přihlašovacích údajů byl obdobný tomu u Spotify. MusicAPI nabízí autentizaci uživatele u jednotlivých streamovacích služeb, ale tu jsem tak jako u Spotify Web API nevyužil. Proces autorizace se tím zjednodušil a k úspěšnému dotázání se na API stačí uvézt Client ID. [63, 64]

Nabídka endpointů je velice omezená. To může být způsobeno jak novostí tohoto API, tak potřebou tohoto API sjednotit dané dotazy tak, aby šly provézt na všech službách naráz. Jediný endpoint, který jsem použil, je /public/search.

/public/search umožňuje vyhledat skladby, alba a playlisty napříč službami. Dotaz na tento endpoint musí obsahovat:

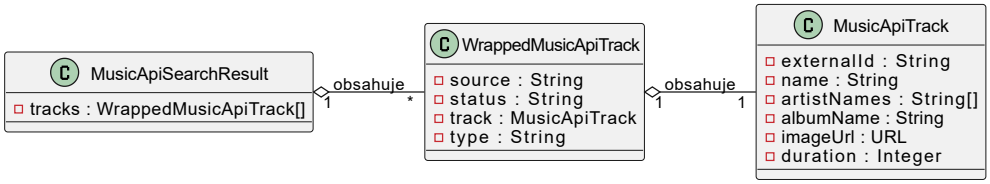
- Název skladby,
- názvy umělců,
- typ – skladba/album/playlist,
- zdroje – u kterých služeb provézt daný dotaz. [65]

```
curl -L -X POST 'https://api.musicapi.com/public/search' \
-H 'Content-Type: application/json' \
-H 'Accept: application/json' \
-H 'Authorization: Token 2cf82db3-4064-4235-8253-16994eb51773' \
--data-raw '{
  "track": "Reload",
  "artist": "Synergy",
  "type": "track",
  "sources": [
    "spotify",
    "soundcloud",
    "applemusic",
    "deezer"
  ]
}'
```

Výpis kódu 4.4 Příklad provedení dotazu na /public/search endpoint v curl

Tento endpoint vrátí od každé zvolené streamovací služby maximálně jednu skladbu, kterou vyhledal. Tato skutečnost je pro mou aplikaci poměrně omezující, jelikož znamená, že za každý dotaz lze nalézt pouze jednu skladbu ze služby Soundcloud. Pokud tedy API nenalezne uživatelem hledanou skladbu rovnou jako tu první, tak vzniká problém. Toho se bez přímého přístupu k Soundcloud API, který momentálně nejde obdržet, nelze zbavit.

Formát vrácených skladeb je službou MusicAPI definovaný, ale od formátu u Spotify Web API se značně liší a mnoho údajů z důvodu rozdílnosti využitých API může chybět. Diagram tříd, které jsem k reprezentaci skladby z MusicAPI využil, je na obrázku 4.5:



Obrázek 4.5 Diagram tříd k MusicAPII

4.2.3 Komunikace s API

Proces komunikace s API jsem si rozdělil na sadu dílčích úkolů, kde za každý úkol má zodpovědnost nějaká třída. Úkoly jsem rozdělil následovně:

4.2.3.1 Ukládání konstant

Pro lepší přehlednost a organizaci bylo žádoucí vytvořit centralizované úložiště konstantních údajů týkajících se API. Jde o údaje jako URL adresy jednotlivých endpointů a v případě MusicAPI i využívané služby. Při implementaci jsem se inspiroval článkem z publikační platformy Medium – [66].

Tuto roli plní v mé aplikaci výčtový typ (enum) APIConstants. Přehledně shromažďuje veškeré potřebné údaje. Pro účely rozšiřitelnosti odpovídají jednotlivé enumy obsažené v APIConstants protokolu APIConstant. Dále také odpovídají protokolu RawRepresentable, díky kterému lze k údajům přistupovat přehlednějším kusem kódu.

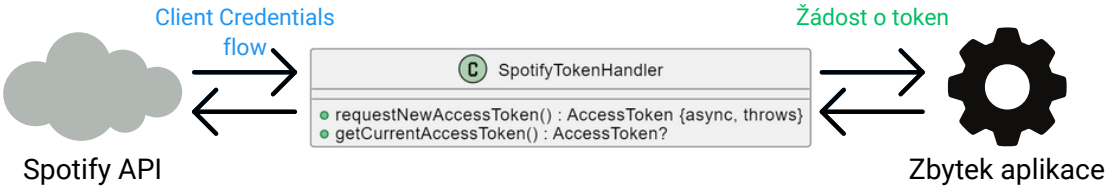
```
enum APIConstants {
    enum Spotify: RawRepresentable, APIConstant {
        static let baseURL = URL(string: "https://api.spotify.com/v1")!
        static let tokenURL = ...
        ...
        case searchTracks
        case ...
        ...
        var rawValue: String {
            switch self {
            case .searchTracks: return "search"
            case ...
            ...
            }
        }
    }

    enum MusicAPI: RawRepresentable, APIConstant {
        static let baseURL = URL(string: "https://api.musicapi.com")!
        ...
        static let sources = ["soundCloud", "appleMusic", "tidal",
                              "youtube", "boomplay", "deezer"]
        ...
    }
}
```

Výpis kódu 4.5 Enum APIConstants

4.2.3.2 Získání přístupového tokenu

Spotify Web API při každém dotazu vyžaduje, aby se dotazující ověřil svým přístupovým tokenem, který získá při autorizaci. Já jsem zvolil autorizační flow Client Credentials, takže jsem implementoval třídu, která se s pomocí této flow bude o správu tokenu starat. MusicAPI žádnou takovou autorizaci nevyžaduje.

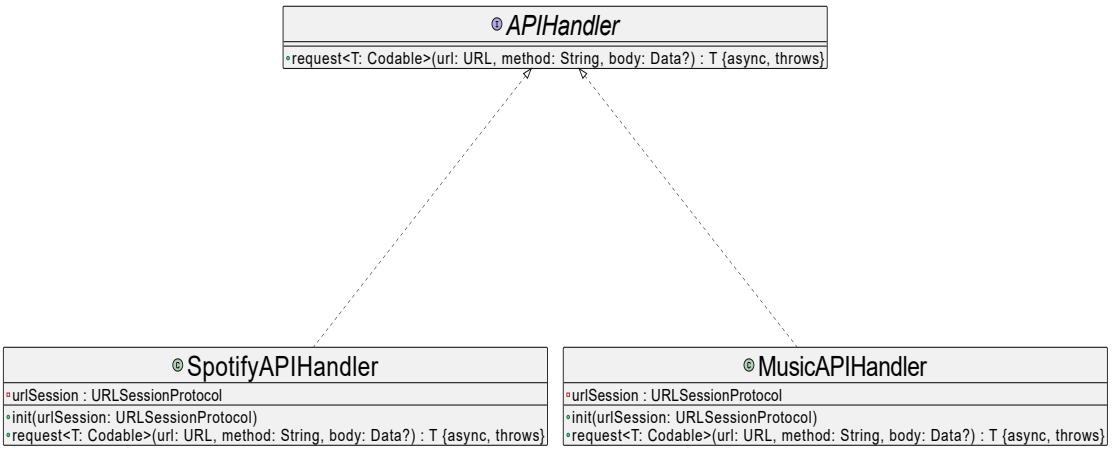
O správu tokenů ke Spotify Web API se stará třída SpotifyTokenHandler. Umožňuje dvě věci – poskytnout aktuální token a zažádat si o nový token pomocí Client Credentials flow.



Obrázek 4.6 Ilustrace SpotifyTokenHandleru

4.2.3.3 Dotazování se na API

Bylo vhodné vytvořit třídy určené specificky k zasílání požadavků na jednotlivá API. Vznikly tak třídy SpotifyAPIHandler a MusicAPIHandler. Tyto třídy mají jedinou funkcionalitu a tou je tvorba požadavku na API a obdržení odpovědi. K síťové komunikaci existuje v jazyce Swift třída URLSession, kterou tyto handlery využívají.

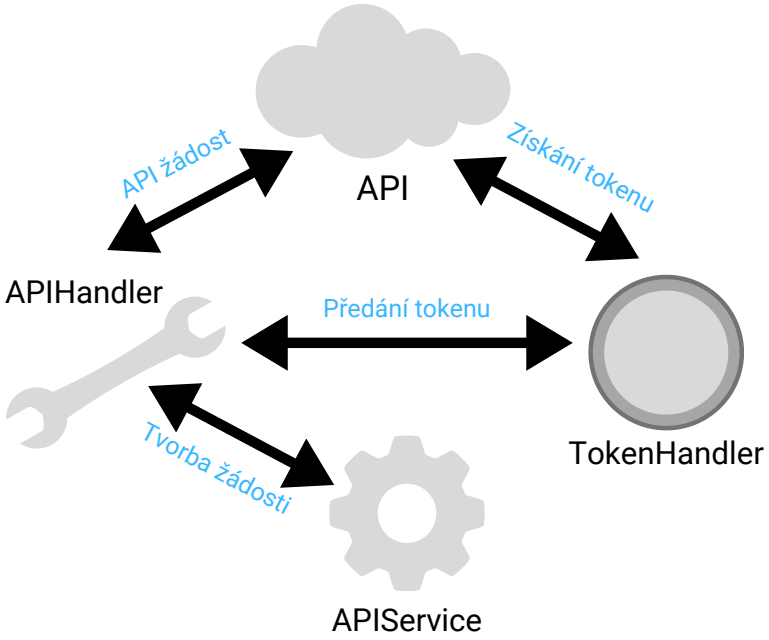


Obrázek 4.7 Diagram APIHandler tříd

4.2.3.4 Manipulace s údaji z API

Za účelem oddělení vrstvy získávající data od vrstvy, která tato data zpracovává, jsem vytvořil třídy SpotifyService a MusicApiService. Tyto třídy fungují jako nadstavby nad APIHandlery a zajišťují transformaci dat získaných z API do formy, kterou využívá zbytek aplikace. S těmito třídami pak komunikují jednotlivé ViewModely, když potřebují vyhledat nějaké skladby.

Reprezentace skladeb se u jednotlivých API liší, a proto se tyto třídy také starají o správný převod API reprezentací skladeb do sjednocené reprezentace, která se využívá napříč aplikací. Tuto reprezentaci nabývá třída Track, kterou lze vidět na obrázku 4.1.

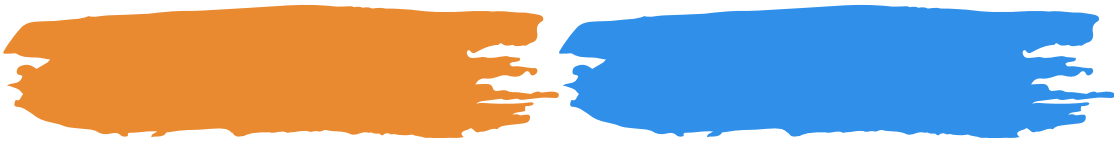


Obrázek 4.8 Ilustrace celé komunikace s API

4.3 Obrazovky

V této sekci se budu zabývat popisem jednotlivých obrazovek, které jsem pro svou aplikaci vytvořil. Budu popisovat jak vizuální stránku obrazovek, tak i funkcionality, které obrazovky nabízí.

Obrazovky jsem se rozhodl tvořit všechny na šířku, jelikož ta nejdůležitější z nich – obrazovka interaktivní úpravy tracklistu – je na šířku a chtěl jsem dodržet konzistentní design. Zároveň SwiftUI momentálně nepodporuje změnu orientace pro jednotlivé obrazovky – lze nastavit pouze globální orientaci pro celou aplikaci. Jako ústřední barvu aplikace jsem si vybral oranžovou barvu s hexa kódem #E98930 a jako komplementární barvu k ní modrou s hexa kódem #3090E9.

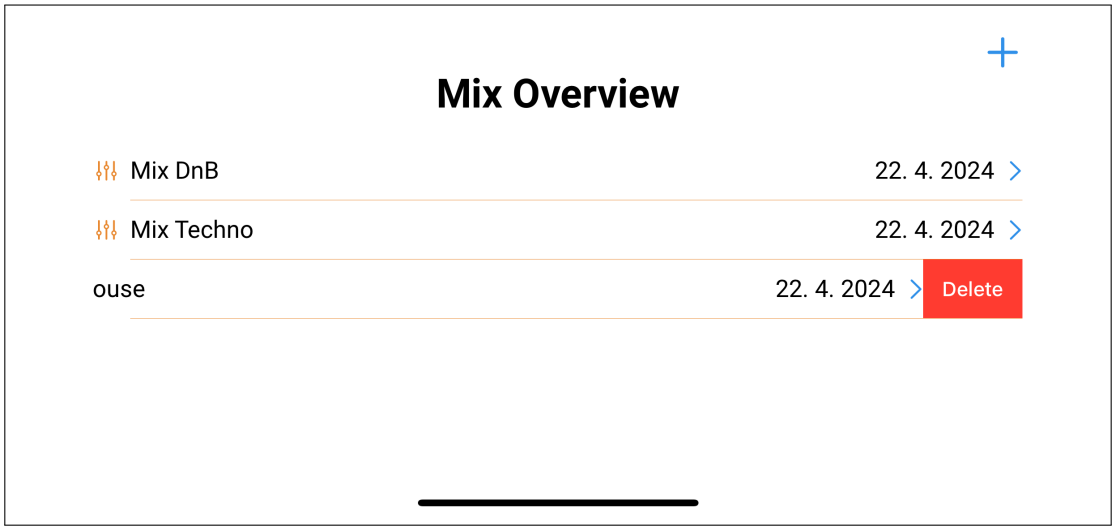


Obrázek 4.9 Vybrané barvy pro mou aplikaci

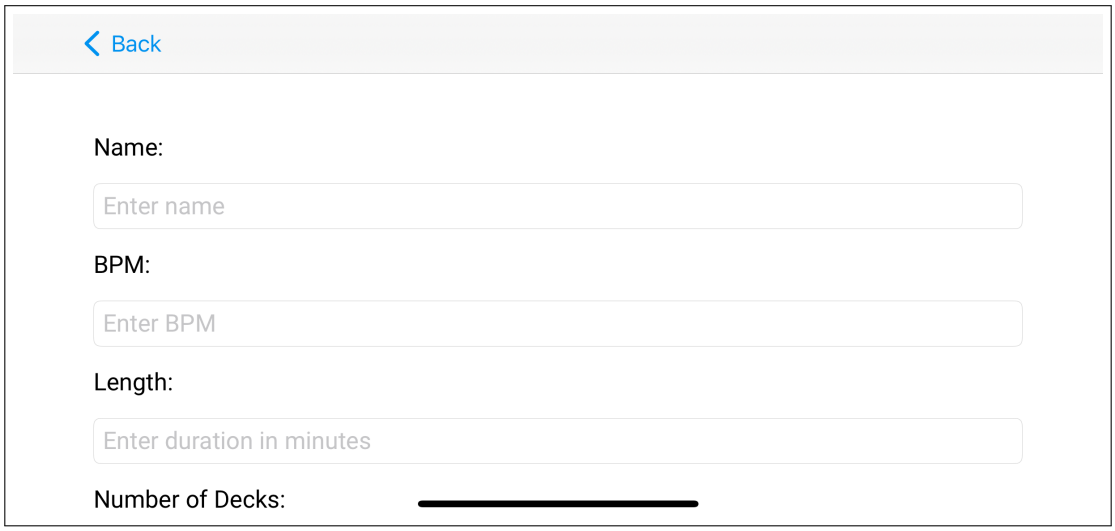
4.3.1 Přehled tracklistů

Obrazovka přehledu tracklistů, neboli Mix Overview, je první obrazovkou, s kterou se uživatel po otevření aplikace setká. Obsahuje seznam vytvořených tracklistů a tlačítko pro vytvoření nového. Zároveň lze tracklisty smazat přejetím prstu doleva a ťuknutím na tlačítko Delete.

Po ťuknutí na tlačítko s ikonkou plusu je uživatel přesměrován na obrazovku, v které uvede potřebné informace o novém tracklistu jako jeho název, BPM a délku. Počet playerů (Number of Decks) je nastaven vždy na čtyři. Na spodku obrazovky poté může kliknout na tlačítko Create, které tracklist vytvoří.



Obrázek 4.10 Obrazovka přehledu tracklistů



Obrázek 4.11 Obrazovka přidání tracklistu

4.3.2 Interaktivní úprava tracklistu

Obrazovka interaktivní úpravy tracklistu je srdcem celé aplikace. Děje se na ní veškerá funkcionalita ohledně přidávání, odebírání, přemisťování a upravování skladeb napříč playery v daném tracklistu. Obrazovka je horizontálně scrollovatelná, což znamená, že se po ní může uživatel vodorovně posouvat. Tím je umožněno zachytit celý tracklist v jediné obrazovce.

Základ obrazovky tvoří tři identické časové osy. Každý indikátor na ose indikuje čtyři takty. Indikátor šestnácti taktů je dvakrát vyšší. Na časových osách je dále indikováno každých deset minut. Z obou stran každé časové osy jsou volné prostory, které lze zaplnit skladbami. Každý z těchto čtyř prostorů symbolizuje jeden player. Na pravé straně k nim se nachází ikony plusu, přes kterou může uživatel vyhledat a přidat skladbu na vybraný player.

V závislosti na způsobu přidání a délce se skladba zobrazí na playeru v podobě buňky. Tato buňka obsahuje název skladby a umělců, časy ve skladbě, v kterých ji zahrát a vypnout, a titulní

fotku. Uvedené časy jsou vyjádřeny v řádu minut a sekund a v řádu taktů. Pokud byla skladba vložena manuálně bez užití API, tak se jako titulní fotka použije ikona disku. Dále se také titulní fotka nezobrazuje, pokud je skladba moc krátká. Toto rozhodnutí jsem udělal z důvodu přehlednějšího zobrazení informací o názvech skladby a umělců. Ze stejného důvodu jsem také pro zmíněné informace implementoval jezdící text v případě nedostatečně dlouhé buňky.

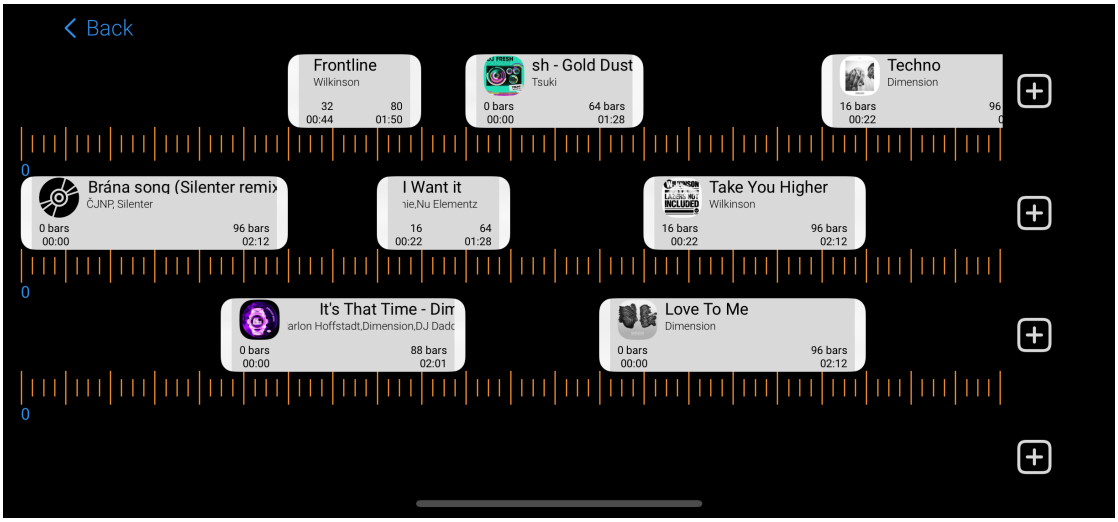
S buňkou skladby lze provádět následující interaktivní akce:

Změna začátku přehrávání Uživatel může podržet a potáhnout vodorovně za levý okraj buňky, který je oproti zbytku buňky světlejší. Potáhnutím doprava či doleva zkrátí, resp. prodlouží, čas ve skladbě, v který ji plánuje začít přehrávat.

Změna konce přehrávání Platí analogicky k prvnímu bodu, jenom pro pravý okraj buňky a s účinkem zkrácení/prodloužení konce přehrávání skladby.

Přemístění skladby Po podržení delším jak sekundu kdekoliv na povrchu buňky kromě jejích okrajů se začne buňka přesouvat stylem Drag & Drop. Uživatel ji může přetáhnout na kterékoliv dostatečně dlouhé volné místo i napříč playery.

Vymazání skladby Uživatel může buňku kompletně vymazat dvojitým ťuknutím kdekoliv na povrchu.



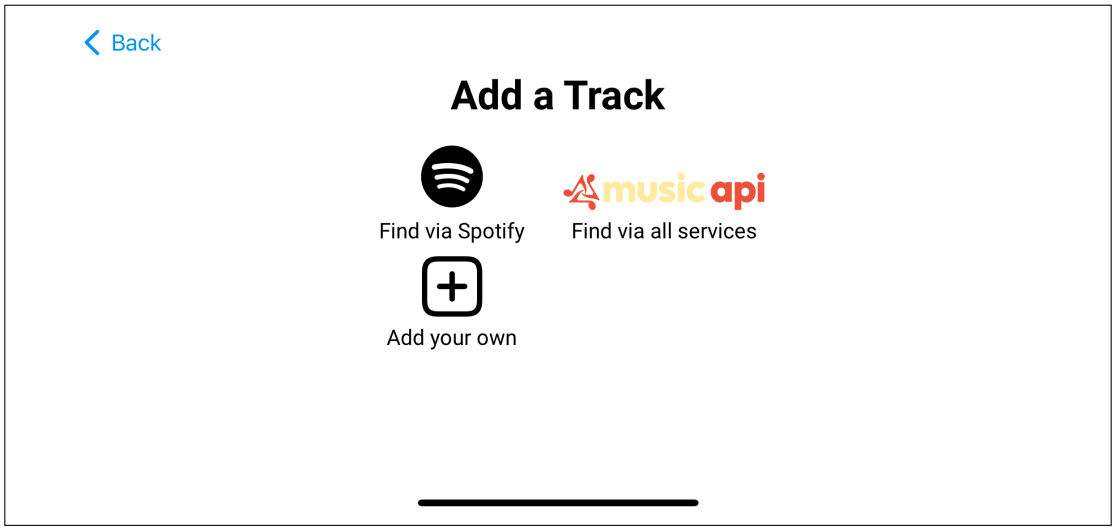
Obrázek 4.12 Obrazovka interaktivní úpravy tracklistu

4.3.3 Přidání skladby

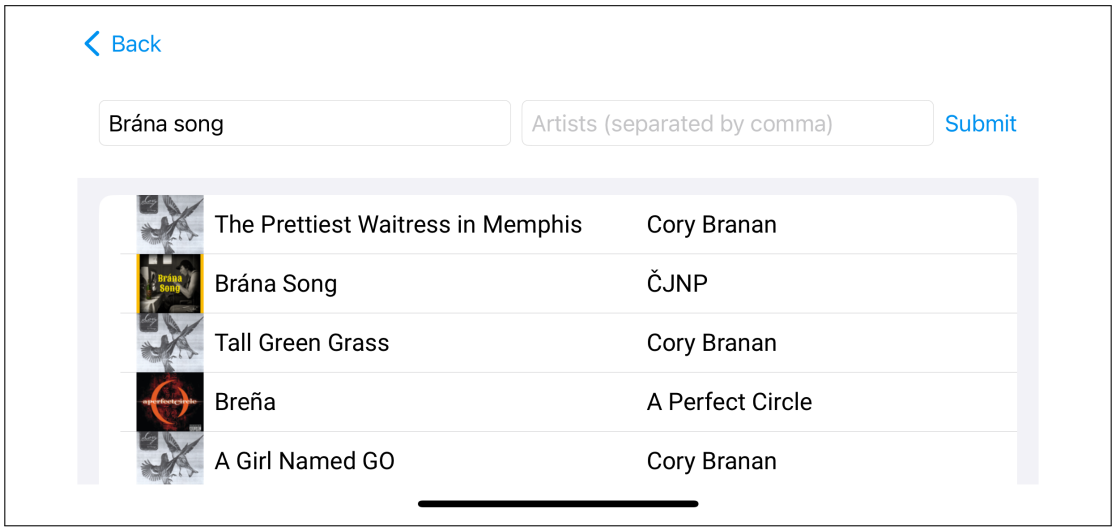
S procesem přidání nové skladby do tracklistu se pojí tři obrazovky. Na první z nich si uživatel vybere způsob vyhledání dané skladby. Má na výběr mezi využitím Spotify Web API, MusicAPI a manuálním vložením. Při zvolení některého z API je uživatel přesměrován na druhou obrazovku, na které může zadat název skladby a názvy umělců. Na obrazovce se mu poté zobrazí seznam až deseti skladeb, které aplikace přes dané API nalezla. Po ťuknutí na některou z nich je převeden na obrazovku třetí.

Na třetí obrazovce může uživatel zkontrolovat a doplnit údaje o přidávané skladbě. Konkrétně se jedná o název skladby a umělců, dobu trvání, BPM, počet sekund mezi úplným začátkem skladby a prvním taktem ve skladbě, čas začátku přehrávání skladby a dobu přehrávání skladby. Na tuto obrazovku je také přesměrován, pokud zvolí možnost manuálního vložení skladby. Po

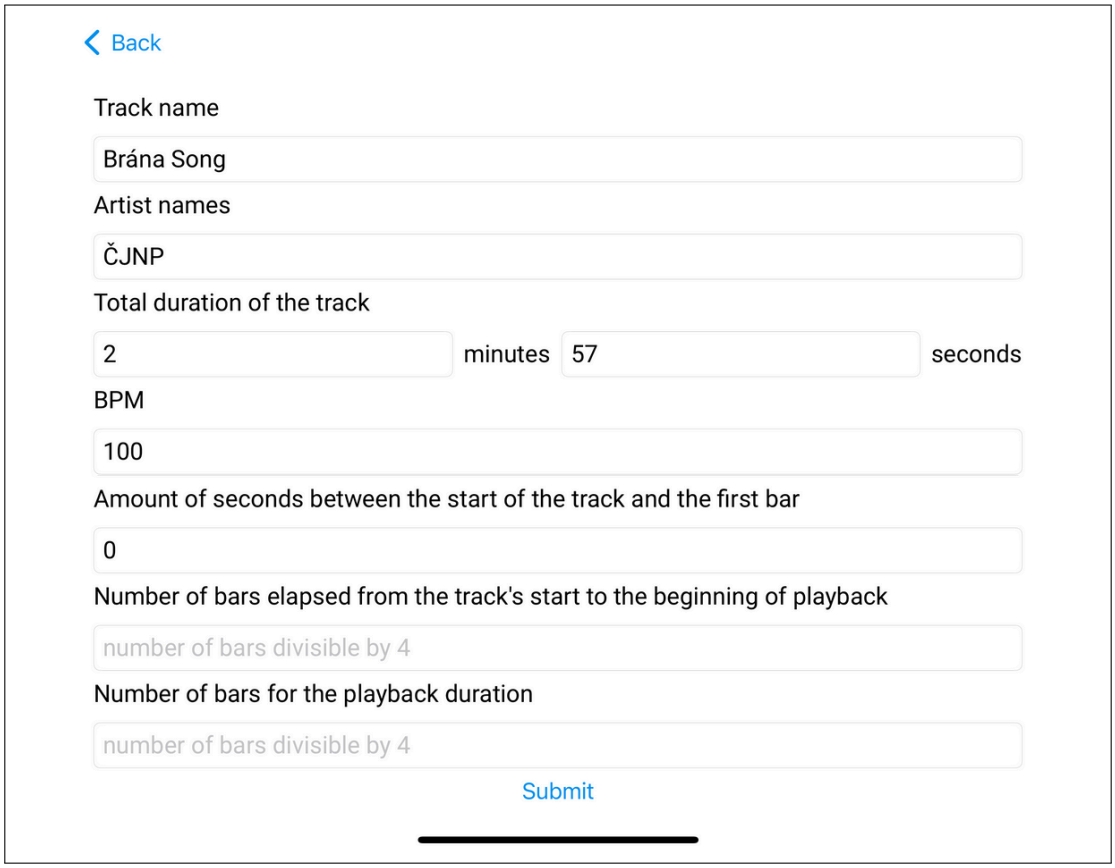
vyplnění všech údajů může skladbu přidat stisknutím tlačítka Submit. Aplikace se stará o validaci vstupu a o nesprávném vstupu uživatele upozorní.



Obrázek 4.13 Obrazovka volby způsobu vyhledání skladby



Obrázek 4.14 Obrazovka vyhledání skladby



Obrázek 4.15 Kompletní obrazovka úpravy informací o skladbě

4.4 Problémy a zajímavosti při implementaci

Je přirozené se při vývoji kteréhokoliv softwaru setkat s řadou komplikací a zábran. Jinak tomu nebylo ani při vývoji mé aplikace. Tomu nepomohl fakt, že technologie SwiftUI a SwiftData jsou mladé a nemají ještě tak rozmanité schopnosti a dokumentaci. V této sekci shrnu několik problémů, s kterými jsem se setkal, a popíšu, jak jsem je řešil, či jestli vůbec vyřešit šly. Zároveň lze některé z uvedených řešení brát jako zajímavosti v kódu.

Se SwiftUI a obecně vývojem mobilních aplikací jsem před touto bakalářskou prací neměl žádné zkušenosti. Proto jsem se přirozeně setkal s mnoha problémy, které ale vycházely z mojí nezkušenosti. O těchto problémech zde mluvit nebudu. Naopak zmíním problémy, které vychází z nedokonalosti frameworků SwiftUI a SwiftData a nebo ze své komplikované podstaty. Takovéto problémy se musely řešit různými alternativními řešeními a nebo se mi vyřešit nepodařily.

4.4.1 Problémy se SwiftData

Práce se SwiftData byla pro mě ze začátku komplikovaná ze dvou důvodů. Prvním z nich byl nedostatek dokumentace a obecně relativně malá online komunita okolo této technologie. Tato skutečnost se dala předpokládat, ale přesto mi přinesla značné komplikace, obzvlášť když jsem se snažil debuggovat problémy spojené s důvodem druhým. Intenzivním vyhledáváním jsem byl ale nakonec schopný se s každým problémem vypořádat.

Druhým důvodem je poměrná nekompatibilita funkcionalit SwiftData s návrhovým vzorem MVVM. Z různých úryvků kódu a funkcionalit, které jsou součástí dokumentace ke SwiftData, je zřejmé, že Apple zamýšlel, aby se skladovaná data udržovala a manipulovala ve View. Tento princip odporuje návrhovému vzoru MVVM, kde by za udržování dat měl mít zodpovědnost ViewModel.

Jako příklad této nekompatibility lze vzít makro @Query, které SwiftData nabízí. Tímto makrem lze označit pole dat, která jsou ukládána v databázi. Kterékoliv manipulace s tímto polem se pak automaticky zapisují i do databáze bez jakékoliv práce navíc. U @Query lze také filtrovat či třídit vytažená data.

Toto makro ovšem nefunguje jinde, než uvnitř nějakého View. ViewModel ani kterákoliv jiná třída zodpovědná za práci s databází přirozeně obrazovkou není, a tak všechny výhody spojené s @Query vývojář ztrácí a o manipulaci s databází se musí postarat sám přes ModelContext.

Samotné modelování entit a následné vkládání (zkušebních) dat pomocí ModelContext bylo taky komplikované, jelikož mi zpočátku aplikace padala ze zdánlivě náhodných důvodů. Po chvíli debuggování jsem zjistil, že u entit se vztahem 1:n záleží, v jakém pořadí se entity inicializují a vkládají do ModelContextu. Pro ilustraci jsem vytvořil následující příklady funkčních a nefunkčních kódů řešící ten samý problém – vložení krabice s dvěma předměty do databáze:

```
// Funguje
let item1 = ItemInBox(name: "item1", box: nil)
let item2 = ItemInBox(name: "item2", box: nil)
let usedBox = Box(name: "box", items: [item1,
                                       item2])
container.mainContext.insert(usedBox)
...
// Funguje
let usedBox = Box(name: "box", items: [])
let item1 = ItemInBox(name: "item1", box: nil)
let item2 = ItemInBox(name: "item2", box: nil)
container.mainContext.insert(usedBox)
container.mainContext.insert(item1)
container.mainContext.insert(item2)
usedBox.items.append(item1)
usedBox.items.append(item2)
```

4.6 Příklady funkčního kódu

```
// Nefunguje
let usedBox = Box(name: "box", items: [])
let item1 = ItemInBox(name: "item1", box: nil)
let item2 = ItemInBox(name: "item2", box: nil)
usedBox.items.append(item1)
usedBox.items.append(item2)
container.mainContext.insert(usedBox)
container.mainContext.insert(item1)
container.mainContext.insert(item2)

// Nefunguje
let usedBox = Box(name: "box", items: [])
let item1 = ItemInBox(name: "item1",
                      box: usedBox)
let item2 = ItemInBox(name: "item2",
                      box: usedBox)
container.mainContext.insert(usedBox)
container.mainContext.insert(item1)
container.mainContext.insert(item2)
```

4.7 Příklady nefunkčního kódu

I přes zmíněné komplikace si dovolím tvrdit, že byl výběr SwiftData oproti Core Data správnou volbou. Například samotné vytvoření databáze a modelování entit dělá v podstatě za vás. Zároveň práce s ModelContext nebyla po prvotním navyknutí velikou překážkou a proces pak již byl přímočarý.

4.4.2 Navigace

Od iOS 16 Apple představil novou strukturu sloužící k navigaci mezi jednotlivými obrazovkami s názvem NavigationStack. Oproti dřívějšímu NavigationView funguje především na deklarativní bázi, což mi činilo značné problémy, obzvlášť když jsem potřeboval navigovat o několik obrazovek zpět. Nabízí i programatický způsob navigace, při kterém si vývojář udržuje zásobník otevřených obrazovek (NavigationPath), kterému může obrazovky odstraňovat a přidávat. [67]

S pomocí NavigationPath a článku pojednávajícím o Router Pattern –[68] jsem byl schopen naimplementovat třídu NavigationRouter, která má zodpovědnost za veškerou logiku ohledně navigace v mé aplikaci. Drží si uvnitř sebe NavigationPath, z které odebírá a do které přidává obrazovky.

```
class NavigationRouter: ObservableObject {
    let modelContext: ModelContext
    ...
    /// Obrazovky na které lze navigovat, včetně jejich parametrů
    enum Destination: Hashable {
        case mixOverview
        case tracklistInfoView(tracklistID: UUID? = nil)
        case ...
    }
    @Published var path: NavigationPath = NavigationPath()
    /// Zkonstruuje obrazovky náležící k jejich Destination
    @ViewBuilder func defineViews(for destination: Destination) -> some View {
        switch destination {
        case .mixOverview:
            MixOverviewView(modelContext: modelContext)
        case .tracklistInfoView(let id):
            TracklistInfoView(modelContext: modelContext, tracklistID: id)
        case .tracklistView(let id):
            ...
        }
    }
    /// Naviguje do dané destinace
    func navigateTo(destination: Destination) {
        path.append(destination)
    }
    /// Naviguje zpět o n obrazovek
    func navigateNBack(n: Int) {
        path.removeLast(n)
    }
    ...
}
```

Výpis kódu 4.8 Třída NavigationRouter

4.4.3 Snap To Grid

Složitější funkcionalitou, kterou jsem pro svou aplikaci chtěl implementovat, je přichytávání skladbových buněk k nejbližšímu indikátoru času. SwiftUI mi s tímto problémem vůbec nepomohlo a musel jsem se obrátit na řešení pomocí manuálního počítání a validování pozic jednotlivých skladeb.

Jako základní jednotku, pomocí které se tyto pozice počítají, jsem zvolil vzdálenost mezi dvěma indikátory sečtenou s šířkou jednoho indikátoru. Tato vzdálenost odpovídá na časomíře čtyřem taktům a v kódu je pojmenovaná jako fourBarsWidth. Myšlenka za tím byla taková, že když se bude každá skladba nacházet na násobku této jednotky, tak se bude nacházet přesně na některém z indikátorů.

Pro výpočty spojené s tímto přichytáváním jsem vytvořil třídu GridHandler. Tato třída nabízí několik dílčích funkcionalit. Je schopna spočítat šířku z počtu barů a naopak pomocí metod getWidthFromBars() a getBarsFromWidth(). Tyto metody pracují s jednotkou fourBarsWidth. Zaokrouhluje pozici, aby odpovídala násobku fourBarsWidth, metodou roundHorizontally(). Zaokrouhlení funguje tak, že se pozice, která je mezi dvěma násobky, zaokrouhlí na bližší z nich.

```
func roundHorizontally(horizontalUnit: CGFloat) -> CGFloat {
    let remainder = horizontalUnit.truncatingRemainder(dividingBy:fourBarsWidth)
    if (remainder < fourBarsWidth / 2) {
        return horizontalUnit - remainder

    } else {
        return horizontalUnit - remainder + fourBarsWidth

    }
}
```

Výpis kódu 4.9 Metoda roundHorizontally()

Dále ověřuje, zda-li se skladba nachází – nebo se plánuje nacházet – na validní pozici. To znamená, že se na daném playeru nepřekrývá s jinou skladbou a že neleží mimo časovou osu. K této validaci slouží metoda validatePosition(), která kromě skladby a playeru bere parametr leadingCoord, který uvádí navrhovanou pozici levého okraje skladby.

```
func validatePosition(track: Track, player: Player,
                      leadingCoord: CGFloat) -> Bool {
    let leadingRounded = roundHorizontally(horizontalUnit:leadingCoord)
    if let tracklist = player.tracklist {
        let tracklistLengthBars = tracklist.durationMinutes.getBars(
                                  bpm: tracklist.bpm,timeUnit: .minutes)
        let tracklistLengthWidth = getWidthFromBars(bars: tracklistLengthBars)
        if (leadingRounded >= tracklistLengthWidth || leadingRounded < 0) {
            return false
        }
    }
    let positionLeft = leadingRounded
    let positionRight = leadingRounded + track.width
    for secTrack in player.tracks! {
        if secTrack.id == track.id {
           continue
        }
        if positionLeft < secTrack.positionRightEdge &&
                          positionRight > secTrack.position {
            return false
        }
    }
    return true
}
```

Výpis kódu 4.10 Metoda validatePosition()

Tato třída a celá funkcionalita přichytávání je využívána ve dvou případech. Prvním z nich je, když uživatel zatáhne za jeden z okrajů skladby za účelem úpravy její délky. Druhým případem je přesouvání skladby z jednoho místa na druhé – Drag & Drop.

4.4.4 Drag & Drop

Již od začátku jsem počítal s tím, že implementování drag & drop funkcionality do mé aplikace bude komplikované. Jako nadějný se jevil protokol Transferable, který umožňuje transportaci objektů napříč obrazovkou. Díky tomu lze jednoduchou a funkcionálně omezenou verzi drag & drop implementovat. Bohužel byl pro mé potřeby tento protokol nedostatečný.

Hlavním důvodem, proč jsem Transferable nemohl použít, byla vizuální podoba celého procesu přetahování skladby. Buňka skladby se po zvednutí uživatelem smrštila a takto zmenšenou ji mohl někam přetáhnout. Pro mou aplikaci bylo toto chování nežádoucí, protože buňka skladby musí mít pořád stejnou podobu, aby uživatel věděl, k jakým indikátorům ji přetahuje a jestli se na volné místo vůbec vejde. Momentálně nikde nešlo nastavit, aby ke smrštění nedošlo. Obecně zatím SwiftUI nenabízí moc možností úpravy práce s tímto protokolem. Tuto skutečnost přisuzuji faktu, že je Transferable protokol mladý.

Drag & Drop jsem tudíž musel implementovat od základu sám. Využil jsem k tomu modifikátor DragGesture, který jsem umístil na buňku skladby. DragGesture umožňuje zachytit a zpracovat gesta přetahování na entitě, na které se nachází. Já jsem podle směru tažení uživatelem upravoval pozici buňky, čímž jsem docílil požadovaného efektu přemisťovaní.

```
struct TrackCell: View {
    @State var viewModel: TrackCellVM
    ...
    var body: some View {
        HStack(spacing: 0) {
            ...
        }
        ...
        .gesture (
            DragGesture(minimumDistance: 0, coordinateSpace: .named("players"))
                .onChanged { gesture in
                    viewModel.drag = gesture.translation
                    if !viewModel.dragging {
                        viewModel.startOfDrag(start: gesture.startLocation)
                    }
                }
                .onEnded { gesture in
                    _ = DragHandler.shared.performDrop(location:
                    CGPoint(x: gesture.location.x - viewModel.xOffset,
                            y: gesture.location.y))
                    viewModel.endDrag()
                }
        )
    }
}
```

Výpis kódu 4.11 Využití DragGesture

O veškerou logiku ohledně puštění buňky se stará třída DragHandler. Tato třída podle informací o pozici puštění, které dodává DragGesture, spočítá, na jaký player a na jaké místo má skladbu zařadit. O dohledání, na který player byla buňka přemístěna, se stará metoda getCorrespondingPlayer().

```
private func getCorrespondingPlayer(location: CGPoint) -> Player? {
    if location.y < 0 {
        return nil
    }
    let trackHeight = UIConstants.shared.track.height
    let indicatorHeight = UIConstants.shared.indicator.big.height
    var lowerBound: CGFloat = 0
    for player in players! {
        lowerBound += trackHeight + indicatorHeight
        if location.y < lowerBound {
            return player
        }
    }
    return nil
}
```

Výpis kódu 4.12 Metoda getCorrespondingPlayer()

O provedení či neprovedení samotného upuštění skladby rozhoduje metoda performDrop(). Tato metoda využívá dříve zmíněný GridHandler k validaci pozice přetahované skladby. Pokud se skladba nachází na volném místě, metoda provede přemístění skladby se všemi nutnými záležitostmi okolo.

```
func performDrop(location: CGPoint) -> Bool {
    if let dragged = draggedTrack {
        let player = getCorrespondingPlayer(location: location)
        if let player = player {
            if (GridHandler.shared.validatePosition(...) {
                let leadingEdgePos = location.x - dragged.width / 2
                let roundedPosition =
                GridHandler.shared.roundHorizontally(...:leadingEdgePos)
                if (dragged.player?.id == player.id) {
                    dragged.position = roundedPosition
                    return true
                } else {
                    dragged.player?.tracks?.removeAll(where: { tr in
                        tr.id == dragged.id
                    })
                    dragged.position = roundedPosition
                    player.tracks?.append(dragged)
                    dragged.player = player
                    return true
                }
            }
    // Všude return false
    ...
}
```

Výpis kódu 4.13 Metoda performDrop()

Přestože se mi podařilo Drag & Drop chování implementovat v podobě, v které jsem chtěl, jedno malé vylepšení se mi implementovat nepodařilo. Momentálně v aplikaci nefunguje, aby

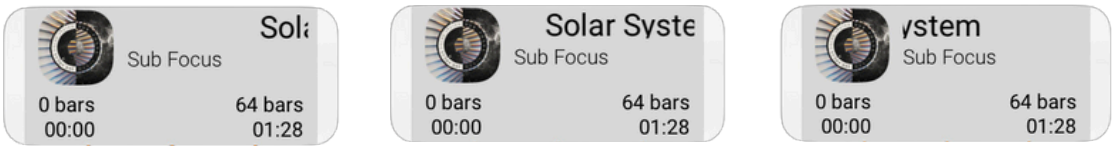
obrazovka sama scrollovala, když uživatel přesouvá skladbu a drží ji u pravého či levého kraje obrazovky. SwiftUI totiž neumožňuje programaticky scrollovat obrazovkou o nějakou vzdálenost. Na tento problém jsem po dlouhé snaze nenašel žádné řešení. Naštěstí pro uživatele existuje způsob, kterým tento problém obejít, a to je skladbu vždy na kraji obrazovky někam odložit, posunout obrazovkou a pak zase skladbu vzít.

4.4.5 Grafická podoba buňky

Uživatelské rozhraní aplikace jsem navrhoval tak, aby se délka buňky odvíjela od délky skladby k ní náležející. Zároveň jsem ale potřeboval, aby se na malou obrazovku mobilního telefonu vešla rozumná část tracklistu. Proto jsem buňky nemohl udělat moc široké. Zároveň jsem je ale nemohl udělat moc úzké, protože by buňky pro skladby s délkou jenom pár barů nebyly vůbec čitelné.

Z grafického návrhu buňky se nakonec stal celkem veliký problém, jelikož byla buňka pro krátké délky skladby opravdu nečitelná. K tomu jsem navíc potřeboval nějak ošetřit případy, kdy byl text pro název skladby nebo umělců delší než daná buňka. K vyřešení obou problémů mě napadlo implementovat jezdící text, který by se spustil, pokud by byl text delší, než buňka umožňuje.

Našel jsem a využil externí knihovnu, která přesně takový text nabízí – [69]. Problém tím považuji za pouze částečně vyřešený, jelikož výsledná podoba buňky pořád není z pohledu uživatele perfektní. Pro účely prototypu aplikace ovšem zatím stačí.



Obrázek 4.16 Buňka s jezdícím textem

4.4.6 Dynamické škálování velikostí

Má aplikace musí fungovat na všech iOS zařízeních majících iOS 17+ a zároveň na zařízeních majících iPadOS 17+. Jednoduše řečeno, měla by fungovat a vypadat obdobně na iPhonech i iPadech. Již v návrhu, konkrétně v sekci o iOS – 3.1.1, jsem uvedl, že vývoj na oba tyto operační systémy zároveň je možný. Tomu tak opravdu bylo – aplikace šla při celém procesu vývoje spustit na iPhone i na iPad. Její vzhled se ovšem značně lišil.

Jako primární zařízení, pro které jsem vyvíjel a na kterém jsem testoval funkcionality a návrhy byl iPhone. Veškeré velikosti jednotlivých komponent, jako jsou buňky, texty a tlačítka, jsem tedy volil tak, aby se tyto komponenty vešly na iPhone. Ve SwiftUI se k určování velikostí využívají hodnoty CGFloat, které se nijak neškálují v závislosti na velikosti obrazovky. Díky tomu byly komponenty na obrazovce iPadu, která je mnohem větší, malé a nečitelné.

Musel jsem tedy najít způsob, kterým dynamicky škálovat uvedené velikosti v závislosti na velikosti obrazovky. Tento úkol nezní moc komplikovaně – stačilo by zjistit rozměry obrazovky a vycházet z nich. Ve SwiftUI k těmto účelům existuje proměnná UIScreen.main.bounds.size. Tato proměnná je ovšem od iOS 16 nepodporovaná, čili někdy v budoucnosti zanikne, a proto bylo žádoucí nalézt jiné řešení, které bude fungovat i v budoucích verzích iOS. [70]

Ke zjištění aktuálních rozměrů obrazovky mi posloužila struktura GeometryReader, kterou SwiftUI nabízí. Funguje tak, že si zabere veškeré možné místo své rodičovské obrazovky a rozměry tohoto místa uvede. Jednoduše použít tuto strukturu na nějaké obrazovce je ovšem zrádné, jelikož její existence může změnit až rozbít rozložení elementů vyskytujících se na obrazovce – layout. Bylo proto vhodné tuto skutečnost nějak obejít.

GeometryReader jsem nepoložil na samotnou obrazovku, ale položil jsem ho do průhledného pozadí této obrazovky. Tím pádem mohl správně uvádět rozměry obrazovky a na prázdném pozadí nemohl narušit žádný layout. Celý proces jsem obalil do funkce useSize(), díky které bylo využití jednoduché a znovupoužitelné. K propagaci informací o změně rozměrů jsem si vytvořil vlastní PreferenceKey, který k těmto účelům slouží. Tento nápad jsem dostal po přečtení článku na fivestars.blog – [71].

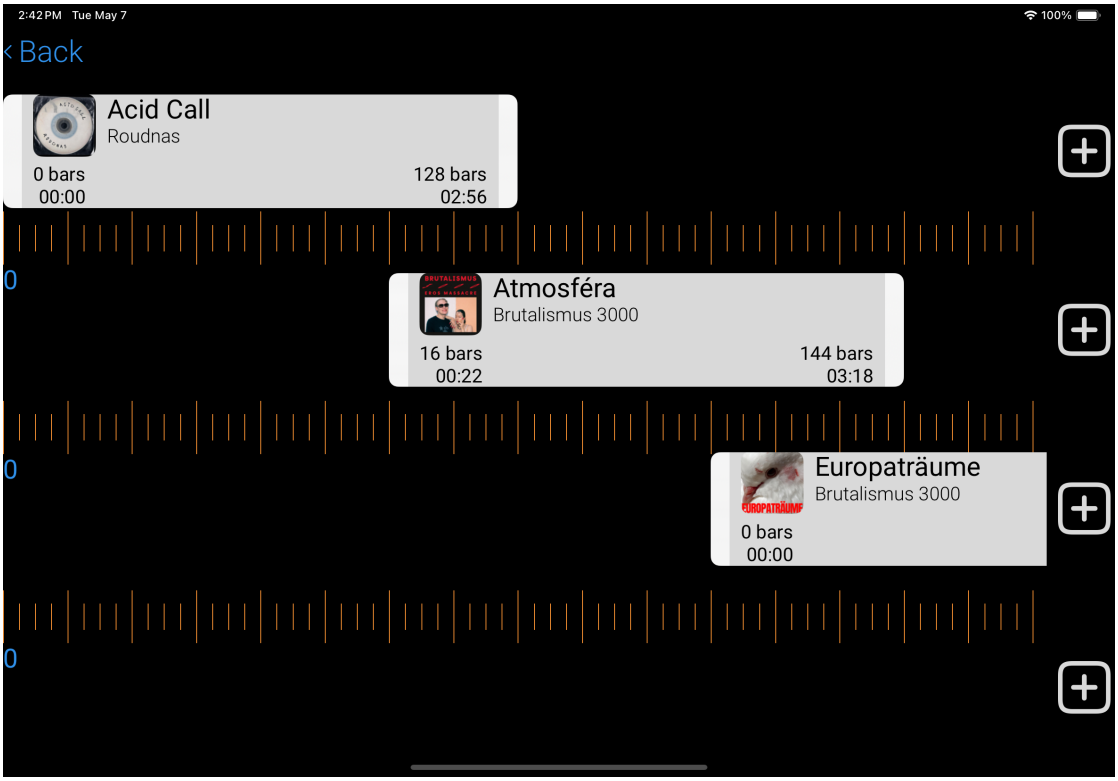
```
extension View {
    func useSize(onChange: @escaping (CGSize) -> Void) -> some View {
        background (
            GeometryReader { geometry in
                Color.clear
                    .preference(key: GeoSizePrefKey.self, value: geometry.size)
                    .onPreferenceChange(GeoSizePrefKey.self, perform: onChange)
            }
        )
    }
}

struct GeoSizePrefKey: PreferenceKey {
    static var defaultValue: CGSize = .zero
    static func reduce(value: inout CGSize, nextValue: () -> CGSize) {
        value = nextValue()
    }
}
```

Výpis kódu 4.14 Metoda useSize() a související GeoSizePrefKey

S rozměry obrazovky se v aplikaci pracuje na několika místech, proto jsem vytvořil uložiště UIConstants. Tuto třídu jsem pojal jako centrální uložiště informací o uživatelském rozhraní, jako jsou velikosti, názvy fontů atd. K určení velikostí často vychází z rozměrů obrazovky, které jsou mu při běhu aplikace dodány.

```
class UIConstants {
    static let shared = UIConstants()
    // Opravdovou velikost se dozví za běhu aplikace
    var screenSize: CGSize = .zero
    struct Indicator {
        var spacing: CGFloat { UIConstants.shared.screenSize.height * 0.02 }
        var width: ...
        ...
    }
    struct Font {
        ...
    }

    struct Track {
        ...
    }
    ...
}
```

Výpis kódu 4.15 Třída UIConstants



Obrázek 4.17 Podoba obrazovky úpravy tracklistu na iPadu Air

Kapitola 5
Uživatelské testování

5.1 Význam testování

Uživatelské testování či testování použitelnosti jsou druhy testování, které mají za cíl vylepšit použitelnost testovaného produktu a odhalit chyby, s kterými se uživatel může při využívání produktu setkat. Tyto testy sdílí pět základních charakteristik:

- Primárním cílem je zlepšit použitelnost produktu,
- účastníci reprezentují opravdové uživatele,
- účastníci provádí opravdové úkoly,
- zaznamenává se, co účastnící udělají a řeknou,
- veškerá data od účastníků se zanalyzují, identifikují se problémy a navrhnou se potřebné úpravy. [72]

Tento druh testování jsem vybral primárně proto, že má moje aplikace relativně omezenou cílovou skupinu a tím pádem bude zpětná vazba respondentů vybraných z této skupiny vysoce relevantní. Na bázi této zpětné vazby budu také schopen navrhnout či rovnou implementovat úpravy, které povedou k vylepšení použitelnosti aplikace.

5.2 TestFlight

Svou aplikaci jsem dle diskuse 3.1.5 nasadil na platformu TestFlight. K využívání této platformy jsem si musel pořídit Apple Developer účet za 2 799 Kč. Distribuce beta verzí aplikace byla se službou TestFlight opravdu jednoduchá a k pozvání nového externího testera, včetně nainstalování aplikace na jeho zařízení, stačilo poslat jediný odkaz. Zároveň jsem díky svému členství v programu Apple Developer schopný pokusit se aplikaci zveřejnit na App Store, což je obchod s aplikacemi podporovaný společností Apple. K této příležitosti jsem vytvořil i ikonu aplikace, která vychází z návrhu časové osy. Aplikaci jsem také pojmenoval jako Tracklistr.

Obrázek 5.1 Ikona mé aplikace Tracklistr

5.3 Cílová skupina a výběr respondentů

Za cílovou skupinu své aplikace považuji DJe, kteří si svůj set plánují předem a hrají celý set hudbu se stejným BPM. Konkrétněji jsem primárně vybíral DJe, kteří hrají žánry Drum & Bass a Techno, protože jsou mi tyto žánry nejbližší a těchto DJů znám více. Samozřejmě také musí mít zařízení s iOS 17+ nebo iPadOS 17+.

Snažil jsem se oslovit jak mezinárodní umělce, tak české umělce. Za nutnou podmínku účasti respondenta jsem považoval to, aby měl alespoň nějakou zkušenost s hraním setu živě. U mezinárodních DJů jsem moc velký úspěch neměl, ozval se mi zpět pouze jeden DJ, který se nakonec neúčastnil testování podle scénářů, ale aplikaci si prošel a navrhl úpravy. Z řad českých umělců jsem sehnal pět respondentů, kteří byli ochotni účastnit se testování.

5.4 Testovací scénaře

K provedení uživatelského testování jsem navrhl sedm vzájemně navazujících scénářů, kde každý scénář odpovídá jedné dílčí funkcionalitě mé aplikace. Každý ze scénářů obsahuje podrobně popsané kroky, kterými se má respondent řídit. Tento přístup jsem zvolil za účelem co největšího zpříjemnění testovacího procesu umělcům vzhledem k jejich nabitému programu.

Testovací scénáře jsem respondentům distribuoval pomocí dotazníku na platformě Google Forms. Ke každému scénáři mohli respondenti uvézt, s čím měli problém, a na konci mohli sdílet své celkové pocity z aplikace. Tento dotazník spolu s odkazem na mou aplikaci jsem zaslal každému respondentovi. Zde je zmíněných sedm scénářů:

5.4.1 Vytvoření tracklistu

Cíl testu: Otestovat, zda-li se správně vytvoří a uloží nový tracklist.

Prerekvizity: Uživatel se nachází na obrazovce přehledu tracklistů.

Průběh:

1. Uživatel klikne na plusovou ikonku, čímž se dostane na obrazovku tvorby tracklistu.
2. Vyplní požadované údaje a zmáčkne tlačítko Create.
3. V nově vytvořeném tracklistu proscrolluje až dozadu, kde ověří, že jeho délka odpovídá zadané délce.
4. Klikne na tlačítko Back, čímž se vrátí zpět na obrazovku přehledu tracklistů. Vytvořený tracklist by se zde měl nacházet. Rozklikne si daný tracklist, aby se dostal zpět na obrazovku úpravy tracklistu.

5.4.2 Přidání skladby přes Spotify API

Cíl testu: Otestovat, zda-li lze jednoduše a efektivně vyhledat, zvolit a přidat skladbu přes Spotify API.

Prerekvizity: Uživatel se nachází na obrazovce interaktivní úpravy tracklistu.

Průběh:

1. Uživatel klikne na plusovou ikonku u libovolného playeru.
2. Vybere si možnost Find via Spotify.
3. Zadá jméno libovolné skladby a volitelně i umělce. Skladba musí být vydaná na Spotify.
4. Ve vytvořeném seznamu skladem vybere tu, kterou hledal. Pokud zde není, tak zkontroluje zadané údaje a popřípadě opakuje vyhledávání s jinou skladbou.
5. U skladby zkontroluje vyplněné údaje a doplní ty zbývající, které záleží na něm.
6. Zmáčkne Submit a zkontroluje, že se skladba správně vložila do tracklistu – je na správném playeru a všechny informace sedí.

5.4.3 Přidání skladby přes MusicAPI

Cíl testu: Otestovat, zda-li lze jednoduše a efektivně vyhledat, zvolit a přidat skladbu přes MusicAPI.

Prerekvizity: Uživatel se nachází na obrazovce interaktivní úpravy tracklistu.

Průběh:

1. Uživatel klikne na plusovou ikonku u libovolného playeru.
2. Vybere si možnost Find via all services.
3. Zadá jméno libovolné skladby a volitelně i umělce. Skladba musí být vydaná na některé ze služeb jako jsou SoundCloud, Apple Music, Tidal, Deezer. Zároveň musí název skladby přesně odpovídat vyhledávané skladbě.
4. Ve vytvořeném seznamu skladem vybere tu, kterou hledal. Pokud zde není, tak zkontroluje zadané údaje a popřípadě opakuje vyhledávání s jinou skladbou.
5. U skladby zkontroluje vyplněné údaje a doplní ty zbývající, které API neposkytlo nebo které záleží na něm.
6. Zmáčkne Submit a zkontroluje, že se skladba správně vložila do tracklistu – je na správném playeru a všechny informace sedí.

5.4.4 Přidání skladby manuálně

Cíl testu: Otestovat, zda-li lze přidat skladbu manuálně uvedením všech potřebných údajů.

Prerekvizity: Uživatel se nachází na obrazovce interaktivní úpravy tracklistu.

Průběh:

1. Uživatel klikne na plusovou ikonku u libovolného playeru.
2. Vybere si možnost Add your own.
3. Vyplní veškeré údaje, které aplikace o skladbě vyžaduje.
4. Zmáčkne Submit a zkontroluje, že se skladba správně vložila do tracklistu – je na správném playeru a všechny informace sedí.

5.4.5 Změna doby hraní skladby

Cíl testu: Otestovat, zda-li lze jednoduše upravit doby hraní skladby z pravé i levé strany.

Prerekvizity: Uživatel se nachází na obrazovce interaktivní úpravy tracklistu a má na ní položenou minimálně jednu skladbu.

Průběh:

1. Uživatel vezme za pravý okraj vytvořené skladby a začne s ním posouvat doprava či doleva. Čas uvádějící konec přehrávání skladby by se měl patřičně upravovat, ale nesmí dojít na to, že by byla celková délka skladby menší než 16 taktů nebo větší než reálná délka skladby. Zároveň by se pravý okraj skladby vždy měl přichytit na nejbližší ukazatel taktů.
2. Uživatel vezme za levý okraj vytvořené skladby a provede to samé, čímž bude upravovat čas uvádějící začátek přehrávání skladby.

5.4.6 Přemisťování skladeb

Cíl testu: Otestovat, zda-li lze jednoduše přemístit polohu skladby na tom samém playeru i mezi různými.

Prerekvizity: Uživatel se nachází na obrazovce interaktivní úpravy tracklistu a má na ní položené minimálně 3 skladby.

Průběh:

1. Uživatel podrží prst na skladbě kdekoliv kromě jejích okrajů. Tím začne skladbu přesouvat.
2. Skladbu přesune na libovolné dostatečně dlouhé volné místo na tom samém playeru. Zkontroluje, že se skladba přichytila k nejbližšímu ukazateli taktů.
3. Skladbu znovu vezme, ale tentokrát ji přesune na jiný player. Zkontroluje, že se skladba přichytila k nejbližšímu ukazateli taktů.
4. Přesune dvě skladby na ten samý player tak, aby mezi nimi nebyl dostatek místa na třetí skladbu. Tu mezi ně zkusí neúspěšně přemístit, přičemž se skladba vrátí na původní místo.

5.4.7 Vymazání skladby a tracklistu

Cíl testu: Otestovat, zda-li lze jednoduše vymazat skladbu i celý tracklist.

Prerekvizity: Uživatel se nachází na obrazovce interaktivní úpravy tracklistu a má na ní položenou minimálně jednu skladbu.

Průběh:

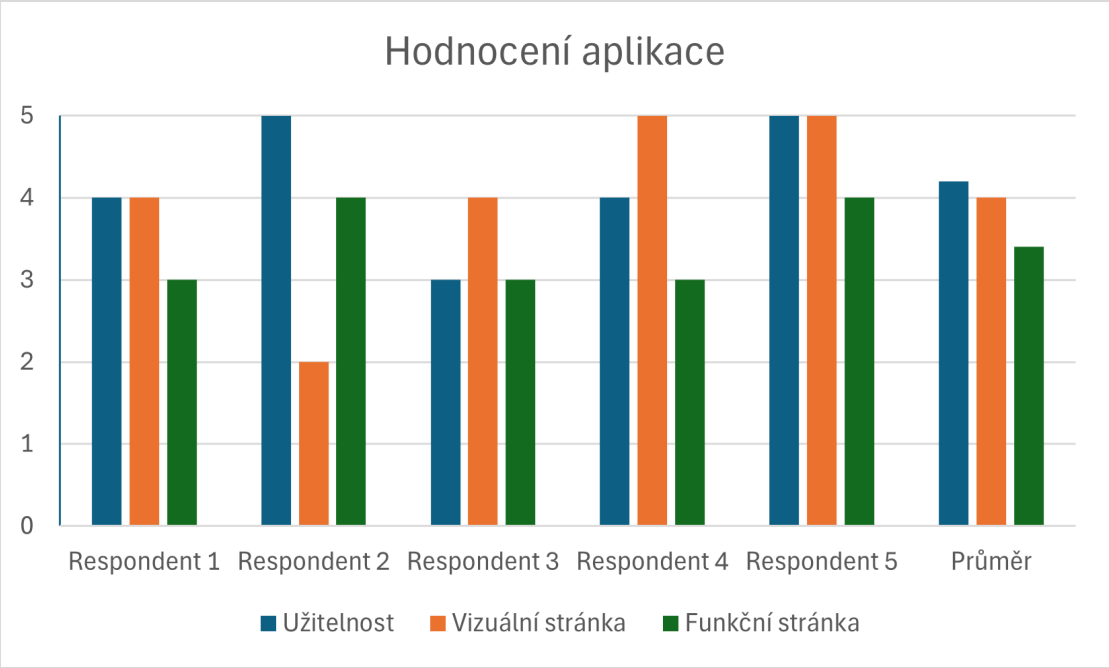
1. Uživatel dvakrát ťukne na skladbu, čímž jí z tracklistu vymaže.
2. Klikne na tlačítko Back, čímž se vrátí zpět na obrazovku přehledu tracklistů.
3. Uživatel na vybraném tracklistu přejede prstem doleva.
4. Klidne na vzniklé tlačítko s nápisem Delete.
5. Zkontroluje, že se tracklist ze seznamu vymazal.

5.5 Výsledky testování

Pro každý z testovacích scénářů jsem vytvořil tabulku, v které jsou uvedené odpovědi všech pěti respondentů. Pokud se scénářem neměl respondent problém, buňka je vyplněna pomlčkou. Kromě scénářů jsou také znázorněny hodnocení použitelnosti aplikace a hodnocení vizuální a funkční stránky aplikace. Tabulky s výsledky testovacích scénářů jsou v příloze – A.

5.5.1 Hodnocení aplikace

Po podstoupení testovacích scénářů mohli DJové ještě ohodnotit užitelnost aplikace, vizuální stránku a funkční stránku. Hodnocení probíhalo na pětibodové škále, kde jeden bod značí špatné hodnocení a pět bodů značí nejlepší hodnocení. V následujícím grafu 5.2 je znázorněno hlasování jednotlivých respondentů pro všechny tři hodnocené kategorie. Dále je pro každou kategorii uveden i celkový průměr.



Obrázek 5.2 Hodnocení aplikace respondenty

5.6 Návrhy a změny vycházející z uživatelského testování

Otestování aplikace cílovou skupinou se ukázalo jako opravdu užitečné. Respondenti v aplikaci odhalili kritické i nekritické chyby, problémy v uživatelském rozhraní a mnoho dalšího. Zároveň poskytli cennou zpětnou vazbu a návrhy na nové funkcionality. Veškeré podněty, a to jak z problémů s testovacími scénáři, tak ze zpětné vazby jednotlivých respondentů, jsem sesbíral a zpracoval. Nyní zde několik z nich zanalyzuji a navrhnu či budu diskutovat vhodné úpravy. Některé úpravy jsem rovnou implementoval.

5.6.1 Kritické bugy

Respondent 1 se setkal se situací, kdy byl schopen roztáhnout buňku skladby dále, než byla celková délka skladby. K tomuto bugu docházelo, když u skladby uvedl, že ji neplánuje hrát od začátku, ale někdy později. Takovouto situaci jsem v kódu při kontrolování vstupu neošetřoval a obratem jsem tento bug opravil.

Respondentovi 3 aplikace spadla, když jím uvedený plánovaný počáteční bod ve skladbě byl později než délka celé skladby. Tento případ jsem zdánlivě při kontrole vstupu ošetřený měl, ale tím, že data s takty ukládám v typu Unsigned Integer a při kontrolování jsem je od sebe odečítal, vznikalo podtečení. Chybu jsem opět ihned opravil, stačilo přemístit porovnávané proměnné tak, aby se už nemuselo nic odečítat.

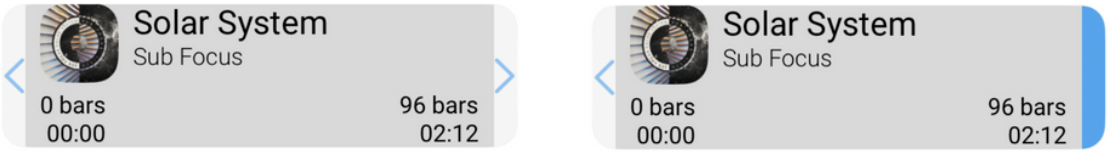
```
// Původně, způsobovalo spadnutí aplikace
if totalDurationBars - currentStartTimeBars < UIConstants.shared.track.minimumBars
// Nově
if totalDurationBars < UIConstants.shared.track.minimumBars + currentStartTimeBars
```

Výpis kódu 5.1 Upravený kód pro práci s Unsigned Integer

5.6.2 Nejasné funkcionality buňky

Mnoho respondentům nebylo zpočátku jasné, jak/že jde s buňkami interagovat. Zároveň při přesouvání či roztahování buňky nedávala aplikace respondentům najevo, kdy už k aktivitě dochází. Před provedením aktivity totiž musí uživatel alespoň sekundu podržet prst na buňce/jejím okraji a aktivita se někdy díky malé obrazovce ani nezaznamená. Toto vedlo ke zmatení a neúspěšným pokusům aktivity provádět.

Z důvodu lepšího znázornění toho, co s okraji buňky může uživatel dělat, jsem přidal na okraje šipky. Zároveň se okraj vybarví, když na něj uživatel podrží alespoň sekundu. Tím pádem je jasnější, k čemu okraje slouží a kdy s nimi lze hýbat. Podobnou funkcionalitu, která uvádí, kdy dochází k aktivitě, by bylo vhodné udělat i na buňce samotné. Například pomocí nějaké animace, vibrace nebo nějakého vybarvení.



Obrázek 5.3 Nová podoba buňky

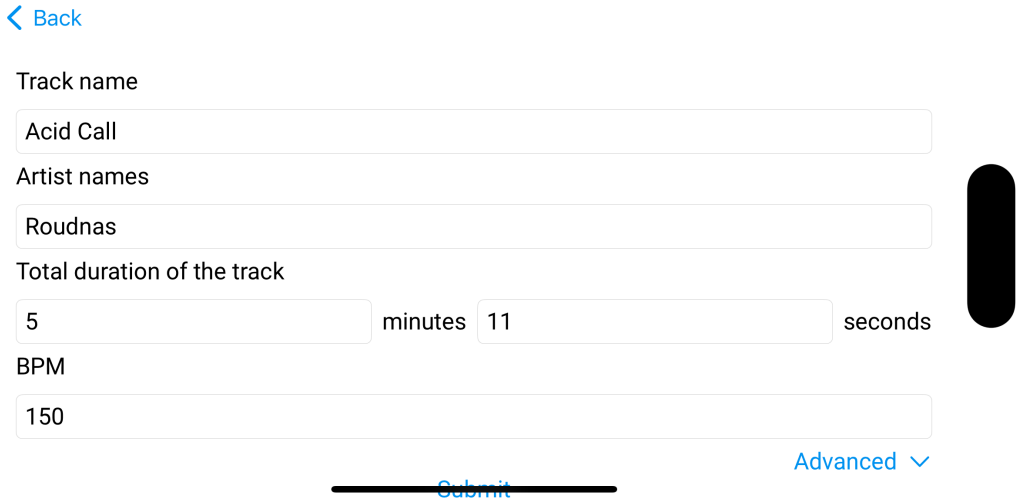
5.6.3 Matoucí názvy parametrů a jejich veliký počet

Někteří respondenti se potýkali s tím, že si s vybranými parametry při procesu doplňování údajů o skladbách nevěděli rady. Názvy jim přišly moc zdlouhavé a málo výstižné. Konkrétně šlo o parametry „Amount of seconds between the start of the track and the first bar“, „Number of bars elapsed from the track´s start to the beginning of playback“ a „Number of bars for the playback duration“.

Tohoto problému jsem se obával, jelikož jsem si sám nebyl jistý, jak tyto parametry trefně nazvat. Vytvořil jsem nejen z těchto účelů grafickou uživatelskou příručku týkající se obrazovky doplnění údajů o skladbě a obrazovky interaktivní úpravy tracklistu. Tato příručka vznikla v angličtině a všichni respondenti ji měli k dispozici. Příručku lze najít v příloze – B

Dále dva respondenti uvedli, že celý proces přidání skladby je zdlouhavý, jelikož u každé skladby musí vyplňovat či kontrolovat výše uvedené tři údaje. S tím souhlasím, a proto jsem vytvořil rozbalující nabídku Advanced, do které jsem tyto parametry přesunul a v které jsou již vyplněné výchozími hodnotami. Tento krok jsem si mohl dovolit, jelikož doba mezi začátkem skladby a prvním taktem je většinou v řádech desetin sekund a tím pádem skoro zanedbatelná. Čas, v kterém skladbu plánuje uživatel začít hrát, a doba přehrávání skladby, jsou oba parametry, které lze později měnit graficky na obrazovce úpravy tracklistu. A upravovat je graficky může být pro uživatele přívětivější, než je vyplňovat ručně.

Proces přidání skladby se tím poměrně zkrátil a například pro skladbu ze Spotify stačí jenom překontrolovat údaje a rovnou zmáčknout tlačítko Submit, jelikož veškerá data už má aplikace vyplněná. Samozřejmě je ale uživatel pořád schopen kterékoliv údaje změnit, včetně těch v nabídce Advanced.



Obrázek 5.4 Nová podoba obrazovky úpravy informací o skladbě

5.6.4 Nekritické bugy v uživatelském rozhraní

Respondent 5 se setkal s vizuálním bugem, kdy mu levý okraj při přetahování divně skákal a chvěl se. Příčinu tohoto bugu se mi nalézt nepodařilo, ale s největší pravděpodobností bude vycházet z funkcionality zaokrouhlování pozic a přichytávání na nejbližší indikátor, kterou vykonává GridHandler. Zároveň se může stát, že se skladba občas při přesunutí přichytí k jinému indikátoru, než ke kterému je nejblíže. Oba tyto bugy nejsou kritické z pohledu funkcionality aplikace, ale byly by dobré identifikovat a spravit.

U buňky může být občas obtížné zachytit její okraje. Tato nepříjemnost je způsobena hned několika důvody. Okraje buněk jsou poměrně úzké a na mobilním telefonu může být těžké je trefit prstem. Zároveň je celá obrazovka scrollovatelná, což znamená, že se někdy může změnit interní zaměření SwiftUI z gesta uživatelského tahání na gesto přetahování po obrazovce za účelem scrollování. Obecně bylo poměrně obtížné implementovat gesta, která lze provádět na buňce uvnitř scrollovatelného prostředí. Vyžadovalo to různá alternativní řešení a často neúspěšné pokusy. Do budoucna by bylo vhodné problematiku prozkoumat ještě více do hloubky a vyzkoušet více různých řešení. Možná by také šlo zkusit celé scrollovatelné prostředí implementovat vlastními silami.

5.6.5 Čistě horizontální orientace aplikace

S čistě horizontální orientací aplikace měli problém respondenti 1 a 2. S respondenty souhlasím, jelikož psaní horizontálně na mobilním zařízení je neobvyklé a může být nepříjemné. O této problematice jsem se zmiňoval už v sekci o obrazovkách – 4.3. Bohužel SwiftUI momentálně nenabízí nativní způsob, kterým lze u jednotlivých obrazovek určovat jejich orientaci. Tento fakt jsem se pokoušel obejít několika způsoby, včetně přechodu k funkcionalitám UIKitu, ale bezúspěšně. Do budoucna je to určitě vylepšení, které by mohlo zpříjemnit uživatelský prožitek.

5.6.6 Budoucí rozšíření

Kromě vyjádření nespokojenosti s provedením některých stávajících funkcionalit mi také několik respondentů navrhlo možné budoucí vylepšení a rozšíření aplikace. Jedním z nich bylo přidání více možností, kterými přidat skladbu. Aplikace by například mohla skladbu načíst ze souborového

systému zařízení. Tuto skladbu by poté zanalyzovala, čímž by zjistila délku, název, ale i tempo a informace o taktech. Aplikace by tedy musela umět pokročile analyzovat audio soubory a na bázi této analýzy odhadnout vlastnosti skladby. Toto zní jako zajímavý problém, kterého řešení bude netriviální a náročné. Jednalo by se ovšem o opravdu užitečné rozšíření aplikace.

Dalším návrhem bylo umožnit DJovi vyznačit ve skladbě jednotlivé její sekce. Tím pádem by viděl, kde je build-up, drop a ostatní části. Poskládat set by pak pro něj bylo jednodušší a hlavně intuitivnější, jelikož momentálně tyto informace musí vědět ze svého DJ softwaru. Takovéto vylepšení by bylo vskutku užitečné a jeho provedení by se nemuselo ukázat jako příliš složité.

Aplikace by také v budoucnu mohla umožňovat plánovat mixy, v kterých se postupem času mění či střídá tempo. Jednalo by se o další velice užitečné rozšíření, které by mohlo oslovit více uživatelů. Implementace a grafické zpracování takovéto funkcionality by se ale mohly ukázat jako celkem komplikované.

Jako poslední rozšíření bych rád zmínil zbylé funkční a nefunkční požadavky, které se v rámci této práce nesplnily. Jedná se o požadavky se střední až nízkou prioritou, jako je sekce nastavení v aplikaci nebo možnost exportu tracklistu do textového formátu.

Kapitola 6
Závěr

Tato bakalářská práce se věnovala především návrhu, implementaci a uživatelskému testování mobilní aplikace na platformy iOS a iPadOS. Cílem této aplikace bylo zjednodušit diskžokejům hraní DJ setu tím, že si set v aplikaci dopředu poskládají a tím si průběh setu naplánují. Při živém přehrávání skladeb jim potom aplikace slouží jako návod, do kterého mohou kdykoli nahlédnout.

V první kapitole byly vytyčeny cíle této práce. V druhé kapitole byla provedena analýza. Nejdříve byl čtenář seznámen s problematikou DJingu a s tím, jak je tato problematika propojena s mou aplikací. Dále byla provedena analýza existujících aplikací řešících tento problém. Z této analýzy vyplynulo, že obdobná aplikace momentálně pro mobilní zařízení neexistuje. Na bázi předchozí analýzy a vytyčených cílů byly sestaveny funkční a nefunkční požadavky, u kterých byla uvedena i jejich priorita.

Třetí kapitola se zaobírala návrhem. Nejprve byly porovnány technologie, s pomocí kterých šlo aplikaci tvořit. Porovnání se týkalo programovacích jazyků, frameworků uživatelského rozhraní, technologií pro persistenci dat a služeb pro uživatelské testování. Z každého druhu technologie byla vybrána ta nejvhodnější. Dále se rozebíraly možné architektury aplikace. Vybrána byla architektura MVVM.

Čtvrtá kapitola se zabývala implementací prototypu aplikace. Nejprve byla popsána práce s databází. Následoval výběr hudebních API a popis komunikace s těmito API. Popisované procesy a třídy byly podloženy diagramy a úryvky kódu. Následovala diskuse nad návrhem obrazovek včetně jejich realizace. Nakonec byly zmíněny problémy a zajímavé funkcionality, které implementaci komplikovaly, včetně popisu jejich řešení.

Pátá kapitola se věnovala uživatelskému testování. Aplikace byla podrobena testování v rámci cílové skupiny. Pro každou dílčí funkcionalitu byl vytvořen testovací scénář a aplikaci takto otestovalo pět DJů. Jejich zpětná vazba byla zpracována a na její bázi byly navrženy či rovnou realizovány úpravy aplikace. Aplikace byla respondentům dodána pomocí platformy TestFlight, která je zároveň součástí App Store Connect. Aplikace byla tedy připravena i k nasazení na App Store.

Zadání bakalářské práce a vytyčené cíle jsou splněné. Vznikla aplikace s moderním a jednoduchým uživatelským rozhraním, která umožňuje interaktivně si naplánovat nadcházející DJ sety pomocí skladeb, které lze přidat několika rozdílnými způsoby. Zároveň z uživatelského testování vzešlo několik možných budoucích rozšíření, která mohou aplikaci dále vylepšit. Jedná se například o další možnosti přidání skladeb, větší míru uzpůsobení dle potřeb uživatele a úpravy uživatelského rozhraní.

Příloha A

Výsledky průchodu testovacími scénáři

A.1 Vytvoření tracklistu

Respondent 1	Při scrollování obrazovky úplně doprava nebylo vyznačeno posledních 10 minut tracklistu, takže bylo těžké poznat, jestli má časová osa opravdu správnou délku.
Respondent 2	Musel jsem vše psát na vodorovné obrazovce.
Respondent 3	–
Respondent 4	–
Respondent 5	–

Tabulka A.1 Testovací scénář 1 – Vytvoření tracklistu

A.2 Přidání skladby přes Spotify API

Respondent 1	Psaní na vodorovné klávesnici je nepříjemné.
Respondent 2	Zase jsem musel psát vodorovně.
Respondent 3	–
Respondent 4	Pojmenování vyžadovaných parametrů bylo nesrozumitelné, nevěděla jsem co mám vyplňovat. I po přečtení příručky. A těžce se mi z hlavy dopočítávaly bary.
Respondent 5	–

Tabulka A.2 Testovací scénář 2 – Přidání skladby přes Spotify API

A.3 Přidání skladby přes MusicAPI

Respondent 1	–
Respondent 2	–
Respondent 3	–
Respondent 4	U tracku se nevyplnila jeho délka.
Respondent 5	–

Tabulka A.3 Testovací scénář 3 – Přidání skladby přes MusicAPI

A.4 Přidání skladby manuálně

Respondent 1	Byl jsem zmatený tím, co znamená „Number of bars elapsed from the track's start to the beginning of playback“ a dostal jsem matoucí chybnou hlášku.
Respondent 2	–
Respondent 3	Aplikace spadla, když jsem tam vyplnil data, kde délka skladby byla minuta, 160bpm, počátek 64 taktů a čas hraní 64.
Respondent 4	–
Respondent 5	–

Tabulka A.4 Testovací scénář 4 – Přidání skladby manuálně

A.5 Změna doby hraní skladby

Respondent 1	Aplikace mě nechala roztáhnout buňku skladby dále, než je celková délka skladby, pokud jsem uvedl, že ji neplánuji zahrát od nultého baru – tedy od začátku.
Respondent 2	Nebylo hned jasné, že se buňky dají roztahovat.
Respondent 3	Funguje to, ale když jsem změnil délku skladby na pouhých 16 taktů, tak byla buňka moc malá k dalšímu tahání a další úpravy hned nešly.
Respondent 4	Zachytit levý okraj u kraje telefonu bylo obtížnější.
Respondent 5	Dragování zleva se trochu chvělo, ale fungovalo správně.

Tabulka A.5 Testovací scénář 5 – Změna doby hraní skladby

A.6 Přemisťování skladeb

Respondent 1	Je těžké poznat, kdy začalo přemisťování skladby.
Respondent 2	–
Respondent 3	–
Respondent 4	–
Respondent 5	Hodilo by se vyznačit, kdy už jde hýbat.

Tabulka A.6 Testovací scénář 6 – Přemisťování skladeb

A.7 Vymazání skladby a tracklistu

Respondent 1	V dotazníku se mluví o ikoně odpadkového koše, ale v aplikaci je napsáno Delete.[1]
Respondent 2	–
Respondent 3	–
Respondent 4	–
Respondent 5	–

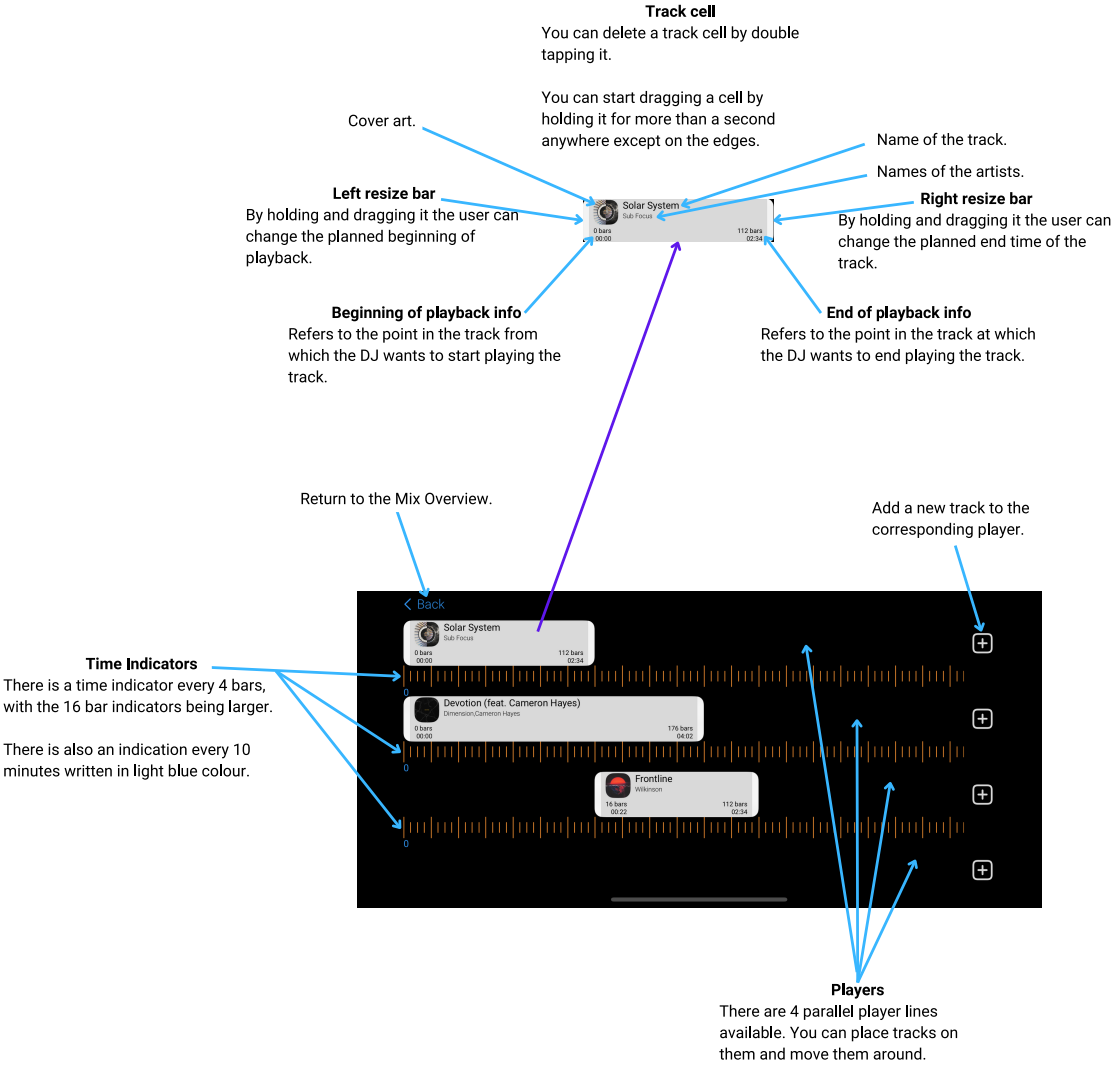
Tabulka A.7 Testovací scénář 7 – Vymazání skladby a tracklistu

[1] Tato chyba byla obratem opravena v dotazníku.

Příloha B

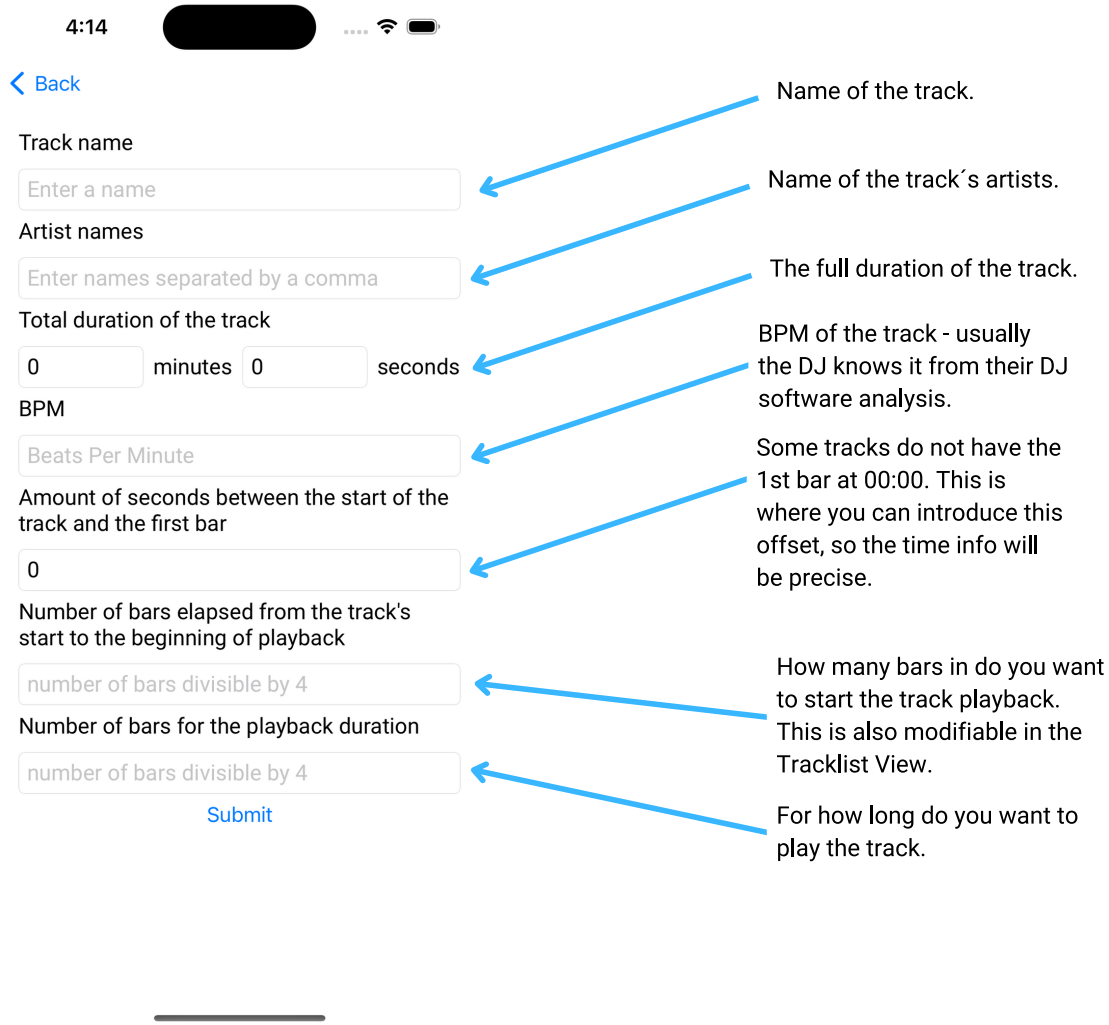
Uživatelská příručka v anglickém jazyce

B.1 Obrazovka interaktivní úpravy tracklistu



Obrázek B.1 Manuál pro obrazovku úpravy tracklistu

B.2 Obrazovka doplnění informací o skladbě



Obrázek B.2 Manuál pro obrazovku doplnění informací o skladbě

Bibliografie

1. *DJ HISTORY* [online]. 2019. Dostupné také z: https://www.skilzdjacademy.com/post/2019/02/01/dj-history. [cit. 25/03/2024].
2. MANGA, Christivie. *Diving Into the History of the Legendary Jamaican Sound System – BLAM UK CIC* [online]. 2022. Dostupné také z: https://blamuk.org/2022/05/12/diving-into-the-history-of-the-legendary-jamaican-sound-system/. [cit. 25/03/2024].
3. *2023-2030 | DJ Equipment Market Research| 102 Pages Report* [online]. 2023. Dostupné také z: https://www.linkedin.com/pulse/2023-2030-dj-equipment-market-research-102-pages-1f. [cit. 25/03/2024].
4. MARK MULLIGAN, MIDiA Research. *IMS Business Report 2023* [online]. 2023. Dostupné také z: https://mcusercontent.com/77ec3e3fadbb08e9f7ddbfb93/files/b95f93cc-316c-433d-9f36-6735c4fc17c7/MIDiA_IMS_Business_Report.pdf. [cit. 25/03/2024].
5. MITTELSTADT, Austin. *The History of DJ Equipment* [online]. 2020. Dostupné také z: https://www.channelaudiogroup.com/single-post/history-of-dj-equipment. [cit. 19/03/2024].
6. DUNWELL, Jack. *Turntables Vs CDJs Vs Controllers: Which Is Better?* [Online]. 2020. Dostupné také z: https://www.musicgiants.com/turntables-vs-cdjs-vs-controllers-which-is-better/. [cit. 19/03/2024].
7. ROSS, Gemma. *Denon DJ announces "the world's most versatile controller"* [online]. 2021. Dostupné také z: https://mixmag.net/read/denon-dj-versatile-controller-lc6000-prime-tech/. [cit. 19/03/2024].
8. *CDJ-3000 - Professional DJ multi player* [online]. 2020. Dostupné také z: https://www.pioneerdj.com/en/product/player/cdj-3000/black/overview/. [cit. 02/04/2024].
9. PROF. ING. VLADIMÍR ŠEBESTA, CSc.; PROF. ING. ZDENĚK SMÉKAL, CSc. Signály a soustavy. In: Fakulta elektrotechniky a komunikačních technologií VUT v Brně, [b.r.], kap. 6. Dostupné také z: https://is.muni.cz/el/1431/podzim2011/Bi5440/um/Signaly_a_Soustavy_BASS.pdf.
10. COTMAN, Andrew. *The Story Behind The First Ever Mixer* [online]. 2015. Dostupné také z: https://stoneyroads.com/2015/05/the-story-behind-the-first-ever-mixer/. [cit. 28/03/2024].
11. *DJM-A9 - 4-channel professional DJ mixer (black)* [online]. 2023. Dostupné také z: https://www.pioneerdj.com/en/product/mixer/djm-a9/black/overview/. [cit. 02/04/2024].

12. GOLDEN, Ean. *The VCI-100 is Dead- Long live the VCI!* [Online]. 2011. Dostupné také z: https://djtechtools.com/2011/01/10/the-vci-100-is-dead-long-live-the-vci/. [cit. 28/03/2024].
13. *DDJ-FLX10 SPLIT. MIX. CREATE.* [Online]. 2023. Dostupné také z: https://www.pioneerdj.com/en/product/controller/ddj-flx10/black/overview/. [cit. 02/04/2024].
14. MGR. MARIE NOVOTNÁ, Ph.D. *Hudební nauka 2* [online]. 2020. Dostupné také z: https://is.muni.cz/el/phil/podzim2020/HV_04/105938489/Hudebni_nauka_2.pdf. [cit. 21/04/2024].
15. MORSE, Phil. Rock the Dancefloor: The Proven Five-Step Formula for Total Djing Success. In: Rethink Press, 2016, s. 111–119. Dostupné také z: https://www.digitaldjtips.com/app/uploads/delightful-downloads/2017/07/Rock-The-Dancefloor-by-Phil-Morse.pdf.
16. MORSE, Phil. *The Global DJ Census 2023* [online]. 2023. Dostupné také z: https://s3.amazonaws.com/fusedesk2/kt144/1529560-6635019d16f7f7.34531069-Digital-20DJ-20Tips-20Global-20DJ-20Census-202023.pdf?AWSAccessKeyId=AKIATKO3JNIJCLNZSK5M&Expires=1715436892&Signature=lolNeMVygcm2d8gcVgzIxfgjc9A%3D&response-expires=1715436892. [cit. 04/05/2024].
17. JONES, Hollin. *What Is a DAW? Digital Audio Workstation Explained | Steinberg* [online]. 2024. Dostupné také z: https://www.steinberg.net/tutorials/what-is-a-daw/. [cit. 20/03/2024].
18. *Creative DJ mixing from your laptop | DJ.Studio* [online]. 2024. Dostupné také z: https://dj.studio/. [cit. 20/03/2024].
19. ALPHATHETA. *Notice of service name changes from Pioneer DJ to AlphaTheta - News - Pioneer DJ News* [online]. 2023. Dostupné také z: https://www.pioneerdj.com/en/news/2023/notice-of-service-name-changes-from-pioneer-dj-to-alphatheta/. [cit. 10/03/2024].
20. ALPHATHETA. *AlphaTheta Corporation acquires Serato Audio Research LimitedExciting acquisition strengthens long-standing partnership between the two leading DJ and audio industry brands - AlphaTheta* [online]. 2023. Dostupné také z: https://alphatheta.com/en/information/alphatheta-acquires-serato-audio-research-limited/. [cit. 13/03/2024].
21. *Cue Points – Serato Support* [online]. 2020. Dostupné také z: https://support.serato.com/hc/en-us/articles/226518228-Cue-Points. [cit. 13/03/2024].
22. *Adding tracks* [online]. 2024. Dostupné také z: https://dj.studio/academy/adding-tracks. [cit. 20/03/2024].
23. DIJKSTRA, Siebrand. *Decks* [online]. 2024. Dostupné také z: https://feedback.dj.studio/p/decks. [cit. 20/03/2024].
24. LARMAN, Craig. Applying UML and patterns: An introduction to object-oriented analysis and design and the Unified Process. In: Prentice-Hall, 2002, kap. 5. Dostupné také z: https://personal.utdallas.edu/~chung/SP/applying-uml-and-patterns.pdf.
25. ROUSE, Margaret. *What is Drag And Drop? - Definition from Techopedia* [online]. 2016. Dostupné také z: https://www.techopedia.com/definition/6918/drag-and-drop. [cit. 19/03/2024].
26. VOLLE, Adam. *IOS | Apple, Updates, Software, & Origin | Britannica* [online]. 2024. Dostupné také z: https://www.britannica.com/topic/iOS. [cit. 02/04/2024].

27. TURNER, Ash. *Number of iPhone Users in the World & USA (Apr 2024)* [online]. 2024. Dostupné také z: https://www.bankmycell.com/blog/number-of-iphone-users. [cit. 02/04/2024].
28. NASH, Axel. *Why iPhone is Better Than Android: iPhone vs. Android:* [online]. 2023. Dostupné také z: https://mobiletrans.wondershare.com/phone-review/why-iphone-is-better-than-android.html. [cit. 02/04/2024].
29. APPLE. *Planning your iPadOS app - Apple Developer* [online]. 2024. Dostupné také z: https://developer.apple.com/ipados/planning/. [cit. 02/04/2024].
30. APPLE. *Swift - Apple Developer* [online]. 2024. Dostupné také z: https://developer.apple.com/swift/%5C#objective-c. [cit. 02/04/2024].
31. HSU, Hansen. *A Short History of Objective-C* [online]. 2017. Dostupné také z: https://medium.com/chmcore/a-short-history-of-objective-c-aff9d2bde8dd. [cit. 02/04/2024].
32. APPLE. *The Objective-C Programming Language Introduction* [online]. 2013. Dostupné také z: https://developer.apple.com/library/archive/documentation/Cocoa/Conceptual/ObjectiveC/Introduction/introObjectiveC.html. [cit. 02/04/2024].
33. THOMASES, Ken. *superset and dynamic runtime | Apple Developer Forums* [online]. 2017. Dostupné také z: https://developer.apple.com/forums/thread/47787?answerId=140496022%5C#140496022. [cit. 02/04/2024].
34. KAZANDJIAN, Sarkis. *Computer Programming Languages: Swift* [online]. 2023. Dostupné také z: https://www.computerscience.org/resources/computer-programming-languages/swift/. [cit. 02/04/2024].
35. APPLE. *Swift - Apple Developer* [online]. 2024. Dostupné také z: https://developer.apple.com/swift/. [cit. 02/04/2024].
36. APPLE. *Extensions | Documentation* [online]. 2023. Dostupné také z: https://docs.swift.org/swift-book/documentation/the-swift-programming-language/extensions. [cit. 02/04/2024].
37. APPLE. *Array | Apple Developer Documentation* [online]. 2024. Dostupné také z: https://developer.apple.com/documentation/swift/array. [cit. 02/04/2024].
38. *SwiftUI vs UIKit: The best choice for iOS in 2024* [online]. 2024. Dostupné také z: https://sendbird.com/developer/tutorials/swiftui-vs-uikit. [cit. 03/04/2024].
39. APPLE. *SwiftUI Overview - Xcode - Apple Developer* [online]. 2024. Dostupné také z: https://developer.apple.com/xcode/swiftui/. [cit. 03/04/2024].
40. APPLE. *Core Data | Apple Developer Documentation* [online]. 2024. Dostupné také z: https://developer.apple.com/documentation/coredata/. [cit. 03/04/2024].
41. APPLE. *SwiftData - Xcode - Apple Developer* [online]. 2024. Dostupné také z: https://developer.apple.com/xcode/swiftdata/. [cit. 03/04/2024].
42. EGGERT, Daniel. *Core Data Overview* [online]. 2013. Dostupné také z: https://www.objc.io/issues/4-core-data/core-data-overview/. [cit. 03/04/2024].
43. APPLE. *Setting up a Core Data stack | Apple Developer Documentation* [online]. 2024. Dostupné také z: https://developer.apple.com/documentation/coredata/setting_up_a_core_data_stack. [cit. 03/04/2024].
44. APPLE. *Meet SwiftData - WWDC23 - Videos - Apple Developer* [online]. 2023. Dostupné také z: https://developer.apple.com/videos/play/wwdc2023/10187/. [cit. 03/04/2024].
45. APPLE. *App Store Connect - Apple Developer* [online]. 2024. Dostupné také z: https://developer.apple.com/app-store-connect/. [cit. 09/04/2024].

46. APPLE. *TestFlight - Apple Developer* [online]. 2024. Dostupné také z: https://developer.apple.com/testflight/. [cit. 09/04/2024].
47. GOOGLE. *Get set up as a tester with App Distribution | Firebase App Distribution* [online]. 2024. Dostupné také z: https://firebase.google.com/docs/app-distribution/get-set-up-as-a-tester?platform=ios. [cit. 09/04/2024].
48. APPLE. *Enrollment - Support - Apple Developer* [online]. 2024. Dostupné také z: https://developer.apple.com/support/enrollment/. [cit. 09/04/2024].
49. PERRY, Dewayne E.; WOLF, Alexander L. Foundations for the study of software architecture. *SIGSOFT Softw. Eng. Notes*. 1992, roč. 17, č. 4, s. 40–44. ISSN 0163-5948. Dostupné z DOI: 10.1145/141874.141884.
50. MARTIN, Robert C. *The Clean Architecture* [online]. 2012. Dostupné také z: https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html. [cit. 10/04/2024].
51. GOSSMAN, John. *Introduction to Model/View/ViewModel pattern for building WPF apps | Microsoft Learn* [online]. 2005. Dostupné také z: https://learn.microsoft.com/cs-cz/archive/blogs/johngossman/introduction-to-modelviewviewmodel-pattern-for-building-wpf-apps. [cit. 10/04/2024].
52. STONIS, Michael; WARREN, Genevieve; JAIN, Tarun; PINE, David. *Model-View-ViewModel - .NET | Microsoft Learn* [online]. 2022. Dostupné také z: https://learn.microsoft.com/en-us/dotnet/architecture/maui/mvvm. [cit. 10/04/2024].
53. UGAYA40. *MVVMPattern - Model–view–viewmodel* [online]. 2012. Dostupné také z: https://upload.wikimedia.org/wikipedia/commons/8/87/MVVMPattern.png. [cit. 10/04/2024].
54. *SoundCloud Application Registration* [online]. 2024. Dostupné také z: https://docs.google.com/forms/d/e/1FAIpQLSfNxc82RJuzC0DnISat7n4H-G7IsPQIdaMpe202iiHZEoso9w/closedform. [cit. 14/04/2024].
55. MUSICAPI. *Supported Music Services* [online]. 2024. Dostupné také z: https://musicapi.com/docs/api-basics/supported-music-services. [cit. 14/04/2024].
56. SPOTIFY. *Web API | Spotify for Developers* [online]. 2024. Dostupné také z: https://developer.spotify.com/documentation/web-api. [cit. 14/04/2024].
57. SPOTIFY. *Authorization | Spotify for Developers* [online]. 2024. Dostupné také z: https://developer.spotify.com/documentation/web-api/concepts/authorization. [cit. 14/04/2024].
58. SPOTIFY. *Client Credentials Flow | Spotify for Developers* [online]. 2024. Dostupné také z: https://developer.spotify.com/documentation/web-api/tutorials/client-credentials-flow. [cit. 14/04/2024].
59. SPOTIFY. *Search for Item | Spotify for Developers* [online]. 2024. Dostupné také z: https://developer.spotify.com/documentation/web-api/reference/search. [cit. 14/04/2024].
60. SPOTIFY. *Get Track | Spotify for Developers* [online]. 2024. Dostupné také z: https://developer.spotify.com/documentation/web-api/reference/get-track. [cit. 17/04/2024].
61. SPOTIFY. *Get Track's Audio Features | Spotify for Developers* [online]. 2024. Dostupné také z: https://developer.spotify.com/documentation/web-api/reference/get-audio-features. [cit. 14/04/2024].
62. HERNAS, Mike. *MusicAPI: What is it & why we built it | MusicAPI.com* [online]. 2023. Dostupné také z: https://musicapi.com/blog/what-is-it-and-why-we-built-it. [cit. 14/04/2024].

63. MUSICAPI. *MusicAPI: Pricing* [online]. 2024. Dostupné také z: https://musicapi.com/pricing. [cit. 14/04/2024].

64. MUSICAPI. *Authorization | MusicAPI.com* [online]. 2024. Dostupné také z: https://musicapi.com/docs/api-basics/authorization. [cit. 14/04/2024].

65. MUSICAPI. *Search | MusicAPI.com* [online]. 2024. Dostupné také z: https://musicapi.com/docs/endpoints/search. [cit. 14/04/2024].

66. HU, Daniel. *Smartly organize API/Endpoints in Swift* [online]. 2022. Dostupné také z: https://medium.com/@hdmdhr/smartly-organize-api-endpoints-in-swift-433d7386d883. [cit. 14/04/2024].

67. APPLE. *Migrating to new navigation types | Apple Developer Documentation* [online]. 2024. Dostupné také z: https://developer.apple.com/documentation/swiftui/migrating-to-new-navigation-types. [cit. 27/04/2024].

68. PALMA, Eric. *Router Pattern for SwiftUI Navigation* [online]. 2023. Dostupné také z: https://www.curiousalgorithm.com/post/router-pattern-for-swiftui-navigation. [cit. 27/04/2024].

69. ZENG, HaiHan. *Marquee* [online]. GitHub, 2021. Dostupné také z: https://github.com/SwiftUIKit/Marquee. [cit. 27/04/2024].

70. APPLE. *main | Apple Developer Documentation* [online]. 2024. Dostupné také z: https://developer.apple.com/documentation/uikit/uiscreen/1617815-main. [cit. 03/05/2024].

71. ZANETELLO, Federico. *How to read a view size in SwiftUI | FIVE STARS* [online]. 2020. Dostupné také z: https://www.fivestars.blog/articles/swiftui-share-layout-information/. [cit. 03/05/2024].

72. DUMAS, J.S.; REDISH, J. A Practical Guide to Usability Testing. In: Intellect, 1999, kap. 2, s. 22–25. Human/computer interaction. ISBN 9781841500201. Dostupné také z: https://www.jedbrubaker.com/wp-content/uploads/2013/03/Dumas-99.pdf.

Obsah příloh

- `readme.txt` ..... stručný popis obsahu média
- `src`
  - `impl.zip` ..... zdrojové kódy implementace
  - `thesis.zip` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF
- `media` ..... adresář se záznamy obrazovky