FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání bakalářské práce

Název:	Mobilní aplikace pro výuku hudby
Student:	Václav Kobera
Vedoucí:	Ing. Jiří Borský
Studijní program:	Informatika
Obor / specializace:	Webové a softwarové inženýrství, zaměření Softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	do konce letního semestru 2023/2024

Pokyny pro vypracování

Cílem práce je navrhnout pomocnou aplikaci pro učitele hudby (např. ZUŠ) pro motivaci svých studentů k cvičení hry na hudební nástroj a rozvíjení hudebních schopností mimo hodiny s učitelem. Aplikace bude navržena pro mobilní zařízení s operačními systémy Android.

Následujte tyto kroky:
· Zanalyzujte požadavky cílových uživatelů pro danou problematiku a existující konkurenční řešení.
· Na základě provedené analýzy navrhněte funkcionality aplikace.
· Proveďte softwarový návrh aplikace.
· Na základě návrhu vytvořte prototyp Androidové aplikace a popište použité technologie.
· Popište možný vývoj aplikace a možná budoucí vylepšení.

Elektronicky schválil/a Ing. Michal Valenta, Ph.D. dne 7. prosince 2022 v Praze.

Bakalářská práce

MOBILNÍ APLIKACE PRO VÝUKU HUDBY

Václav Kobera

Fakulta informačních technologií
Katedra softwarového inženýrství
Vedoucí: Ing. Jiří Borský
11. května 2023

Obsah

Seznam obrázků

Seznam výpisů kódu

Chtěl bych poděkovat J. Borskému za mentorskou činnost při tvorbě práce. Dále také T. Kalvodovi za pomoc při práci s matematickými koncepty při práci s analýzou zvuku a T. Koberové za asistenci při zpracování textové části práce.

Prohlášení

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

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 2373 odst. 2 zákona č. 89/2012 Sb., občanský zákoník, ve znění pozdějších předpisů, tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené.

V Praze dne 11. května 2023 .....

Abstrakt

Tato bakalářská práce se zabývá návrhem a vývojem prototypu mobilní aplikace pro zařízení s operačním systémem Android, která má povzbudit mladé houslisty ke cvičení mimo pravidelné výukové hodiny s učiteli. Klíčovou součástí aplikace je implementace analýzy zvuku, která nabízí zpětnou vazbu o výkonu uživatele v reálném čase. Dále obsahuje vykreslení notového zápisu na zařízení Android. Prototyp obsahuje ladičku, metronom, stupnice a hru zaměřenou na rozpoznání not v notovém zápise stupnice a připravenou část pro budoucí galerii skladeb. Tento projekt slouží jako případová studie návrhu a implementace mobilní aplikace přizpůsobené specifickým potřebám mladých houslistů a demonstruje potenciál technologií pozitivně ovlivnit jejich výuku.

Klíčová slova mobilní aplikace, Android, hra na housle, Kotlin, zpracování zvuku, vykreslení notového zápisu, diskrétní Fourierova transformace, Jetpack Compose

Abstract

This bachelor thesis deals with the design and development of a prototype mobile application for Android devices to encourage young violinists to practice outside of regular lessons with their teachers. A key component of the app is the implementation of sound analysis, which offers real-time feedback on the user's performance. It also includes a rendering of sheet music on Android devices. The prototype includes a tuner, metronome, scales, and a game focused on recognizing notes in scale notation, and a ready-made section for a future song gallery. This project serves as a case study for the design and implementation of a mobile application tailored to the specific needs of young violinists and demonstrates the potential of technology to positively impact their learning.

Keywords mobile application, Android, violin playing, Kotlin, sound processing, rendering of musical notation, Discrete Fourier Transform, Jetpack Compose

Seznam zkratek

12-ET	12-ti tónový rovnoměrně temperovaný styl pro reprezentaci tónů. 19, 36
API	aplikační rozhraní pro programování aplikací. 11, 12, 15, 18, 35
BOM	způsob verzování více knihoven najednou. 18
BPM	jednotka určující počet úderů za minutu. 39
CI	metoda, která automatizuje jednotlivé fáze vývoje aplikací. (pod-část CI/CD). 39
CI/CD	metoda, která automatizuje jednotlivé fáze vývoje aplikací.. 31
DFT	diskrétní Fourierova Transformace. 11, 20–23, 34
DI	programovací návrhový vzor pro vkládání závislostí. 16, 18, 33, 35
DP	grafická jednotka k určení velkosti objektů určená konkrétním zařízením (rozdílné od pixelů). 14
DSL	Doménově specifický jazyk zaměřen na omezenou, konkrétní problémovou doménu. 17
FFT	rychlá Fourierova Transformace (Fast Fourier Transform). 21, 23, 34, 35
HPS	postup pro zpracování zvuku hudebních nástrojů. 22, 35, 39
IDE	integrované vývojové prostředí. 10, 18
IO	vstupně výstupní operace. 11, 12
JSON	způsob zápisu dat (datový formát) nezávislý na počítačové platformě. 16, 34
KMM	technologie pro vývoj multiplatformních mobilních aplikací. 10, 16
MP3	formát kódování audia, založený na kompresním algoritmu. 39

MVI	druh programovacího návrhového stylu. 16, 17
MVP	druh programovacího návrhového stylu. 17
MVVM	druh programovacího návrhového stylu. 16, 17
NDFT	neuniformní diskrétní Fourierova Transformace (v této práci pouze s neceločíselnými frekvencemi). 23, 35, 40
OS	operační systém. 9
PCM	Audio data zakódovaná v diskrétním časování. 18, 20, 21, 26–28, 35
PX	nejmenší (bezrozměrná) jednotka digitální rastrové grafiky. 14
SQL	strukturovaný dotazovací jazyk, který je používán pro práci s daty v <relačních databázích. 16, 36
TOML	technologie spojená s verzovacími katalogy. 18, 34
UI	Uživatelské rozhraní. 6, 9–12, 16, 17, 40
URI	jednotný identifikátor zdroje složený z textového řetězce. 14
XML	programovací značkovací jazyk. 12, 13

Úvod

Role technologií ve vzdělávání se v posledních desetiletích výrazně vyvinula a vzniklo mnoho aplikací a nástrojů, které podporují a zlepšují výuku. V oblasti hudebního vzdělávání je stále potřeba zapojit a motivovat mladé hudebníky ke cvičení a rozvíjení jejich dovedností nad rámec běžných vyučovacích hodin s učiteli. To platí zejména pro housle, oblíbený, ale náročný nástroj, jehož zvládnutí vyžaduje soustavné cvičení.

Tato bakalářská práce se zabývá vývojem mobilní aplikace pro zařízení s operačním systémem Android, která je speciálně navržena tak, aby motivovala mladé houslisty ke cvičení a věnování času svému nástroji i mimo výuku.

Hlavním cílem tohoto projektu je prozkoumat technické aspekty vývoje mobilní aplikace, která může efektivně podporovat mladé houslisty v jejich samostatném cvičení. Jednou z klíčových součástí aplikace je implementace analýzy zvuku, která umožňuje aplikaci poskytovat zpětnou vazbu o kvalitě výkonu uživatele v reálném čase, což mu pomáhá zdokonalovat techniku a zlepšovat jeho dovednosti. Zároveň tato zpětná vazba cílí na udržení pozornosti hudebníka.

Ačkoli hlavní těžiště této práce spočívá v technickém vývoji aplikace, pojednává také stručně o významu samostatné praxe v hudebním vzdělávání a o tom, jakou roli mohou technologie hrát při usnadňování a zlepšování tohoto procesu. Projekt slouží jako případová studie návrhu a implementace mobilní aplikace přizpůsobené specifickým potřebám mladých houslistů a demonstruje potenciál takových nástrojů pozitivně ovlivnit jejich výuku.

Prostřednictvím vývoje a zkoumání této mobilní aplikace si tato bakalářská práce klade za cíl přispět k probíhající diskusi o úloze technologií v hudebním vzdělávání, konkrétně při podpoře a motivaci mladých hudebníků k tomu, aby se věnovali svému nástroji a samostatně cvičili.

Kapitola 1

Analýza základních požadavků aplikace a volba technologie

Tato kapitola se zaměří na průzkum požadavků uživatelů a vlastnosti již existujících aplikací.

V této práci je kladeno za cíl navrhnout a implementovat prototyp aplikace pro systém Android, který by ukázal směr pro možnou budoucí aplikaci uspokojující potřeby hudebních pedagogů i jejich studentů a poskytne interaktivní platformu, která plynule a příjemně propojí teorii a praxi.

Mobilní zařízení jsou dnes často aplikovány ve výuce dětí. Výhodou aplikací na těchto zařízení je, že děti umí zacházet s těmito zařízeními, jelikož tyto zařízení používají běžně a denně. Také mají mobilní zařízení téměř vždy po ruce. Proto je vhodné vytvořit tuto aplikaci pro mobilní zařízení, které je snadno dostupné, jednoduše ovladatelné a známé pro cílovou skupinu uživatelů. [1]

Tato práce si nebere za cíl dostat se přímo do výuky učitelů, ale motivovat žáky ke hraní mimo lekce s učitelem, které je při hraní na hudební nástroj velmi důležité. Avšak pro učitele je to obvykle těžký úkol, přimět hudebníky hrát mimo vymezené hodiny, jelikož děti často zvolí nějakou zábavnější činnost, kterou vyplní svůj volný čas.

1.1 Požadavky pedagogů

Pochopení požadavků uživatelů je klíčové pro vývoj efektivní aplikace, která vyhovuje potřebám hudebních pedagogů a jejich studentů. V této části se zaměří především na obsah aplikace a směr, kterým by se měla ubírat. Aby aplikace motivovala mladé houslisty ke cvičení mimo vyučování a aby byl vytvořen smysluplný obsah aplikace zajišťující její relevanci v reálných výukových scénářích, konzultovali jsme ji se zkušenými pedagogy pracujícími s dětmi ze základních škol a s učiteli uměleckých škol.

1.1.1 Dotázaný č. 1

- věk: 40-50 let
- pohlaví: žena
- profese: Učitelka vyučující hudební nauku na základní škole

Učitelka z praxe tvrdí, že děti mají rády hry založené na rytmu a jejich začlenění do aplikace by efektivně zapojilo mladé hudebníky. Kromě toho navrhuje použít barevně odlišené noty, které by dětem pomohly snadněji rozpoznat notový zápis a pracovat s ním. Vítá hry související s rozpoznáváním zvuků a upřednostňuje aplikaci reagující v reálném čase. Učitelka je optimistická, že tyto aplikace mohou zlepšit proces výuky. Doporučuje také zařadit durové a mollové stupnice, rozpoznávání intervalů, procvičování rytmu a zajistit, aby aplikace reagovala a byla pro děti zábavná. Pro možnou budoucí aplikaci by volila jedině kladné motivační prostředky, myslí si, že by jinak aplikaci samotní hudebníci nepoužívali.

1.1.2 Dotázaný č. 2

- věk: 20-30 let
- pohlaví: žena
- profese: Učitelka hry na flétnu a hudební nauky v ZUŠ

Učitelka uvádí, že existující aplikace, které použila, efektivně nepodporují výuku hry na nástroj, a proto je studenti využívají jen zřídka. Domnívá se, že ideální aplikace by měla propojovat teoretické i praktické aspekty hudební výchovy. Učitelka také navrhuje, že reaktivnější a interaktivnější aplikace by mohla být pro studenty přitažlivější ve srovnání se statickými aplikacemi, které používala doposud. Nemyslí si však, že by se aplikace měla výrazně zaměřovat na výuku techniky hry na nástroj, protože by nemusela být užitečná pro mladé studenty, kteří potřebují více praktického vedení.

1.1.3 Dotázaný č. 3

- věk: 50-60 let
- pohlaví: žena
- profese: Učitelka hry na housle v ZUŠ

Učitelce by vyhovovalo mít aplikaci, která by motivovala hudebníky pro hraní na hudební nástroj pomocí kladných motivačních prostředků. Aplikace, pomocí které by učitel mohl kontrolovat žáky, jí nepřišla vhodná. Tento postoj se snažila podpořit myšlenkou vývojem hudby samotné, kde dnes většina lidí stejně nehledá možnou kariéru, ale spíše jako zdroj zábavy.

Od aplikace by si přála, aby děti dokázala motivovat ke hře na hudební nástroj. Představovala by si v aplikaci snadná cvičení, na kterých se hudebník rozehraje, jelikož často stačí dostat samotné hudebníky k nástroji. Pokud by se aplikací podařilo dostat studentům nástroj do rukou, je velká pravděpodobnost, že by se podívali i na cvičení, která obdrželi na hodině.

Osobně říkala, že technologie asi nedokáží nahradit samotnou výuku v ZUŠ, ale motivovat žáky ke hraní by je aplikace mohla. Čas věnovaný nástroji by vedl k lepším výsledkům studia. Myslí si, že aplikace obsahující teorii hudby, by byla vhodná spíše pro starší uživatele, kteří by měli zájem se naučit něco nového, ale u mladších si nedokázala představit, že by to mohlo fungovat.

1.1.4 Výstupní informace

Učitelky nevidí smysl ve tvorbě aplikace pro mladší uživatele obsahující velké množství informačních zdrojů. Spíše by se měla zaměřit na motivaci studentů vzít samotný nástroj do rukou a zahrát si. S trendem vývoje hudby by měla spíše směřovat k vytvoření pozitivního vztahu hudebníka s nástroji, než drilovat samotnou techniku.

Z rozhovorů bylo zjištěno, že většina aplikací učitelkám při práci moc nepomáhá a hudebníci často tyto aplikace nepoužívají. Myslí si, že většina aplikací necílí na zábavné prvky těchto cvičení, ale spíše na jakousi povinnost tuto aplikaci použít. Tvrdí, že aplikace, která by dokázala kontrolovat hudebníky v reálném čase, by mohla mladé hudebníky zaujmout, a že by je tato aplikace mohla přimět ke hraní na nástroj i mimo hodiny, což by mělo s velkou pravděpodobností pozitivní dopad na jejich vývoj.

1.2 Existující projekty

Analýza existujících projektů bude zkoumat použitelnost aplikací, zda aplikace rozvíjí uživatelovy hudební schopnosti a jejich dostupnost pro mladé uživatele. Hlavním cílem při analýze těchto aplikací bude důraz na samotnou motivaci uživatele hrát na hudební nástroj. Aplikace byly primárně vybírány z Google Play Store. Dále byly vybrány projekty zmíněné v rozhovorech s pedagogy.

1.2.1 Trala

Trala je mobilní aplikace zaměřená na výuku hry na housle. Veškeré funkce aplikace jsou zdarma. Zpoplatněné byly lekce s učitelem, které se dali pomocí aplikace přímo domluvit. Aplikace obsahovala ladičku, u které nebylo možné nastavit výchozí frekvenci a ladění mi přišlo velmi nepřesné. Video lekce v aplikaci byly velmi dobře zpracované v anglickém jazyce i s titulky. Pokud uživatel hledal něco specifického, tak ve videích většinou našel to, co hledal. Součástí této aplikace bylo také hraní podle not, kde aplikace kontrolovala, jestli hráč hraje to, co je v notách, a na konci vyhodnotila, jak se hráči dařilo. Notový zápis skladby bylo možné přepnout na zápis pomocí kostek různých barev umístěných v různých výškách. Tempo při hraní bylo možné upravit v nastavení, ale nebylo zde možné si danou skladbu poslechnout. Po dohrání si uživatel nemohl poslechnout ani jak hrál on sám. Výsledné skóre nebylo moc informativní. Jednalo se pouze o skóre od 0 do 6 ti, které uživateli nedokázalo přesně určit kde chybuje.[2]

Výsledné skóre nebylo nikde jinde propagováno, ani zde nebyly žádné statistiky, kde by se člověk mohl dozvědět jak postupuje. Samotné hraní nebylo moc motivující, ale skladba se z této části naučit dala.

Přišlo mi, že aplikace cílí spíše na starší uživatele, kteří se chtějí naučit hrát na housle. Jednotlivé hraní s předlohou mi nepřišlo moc motivující spíše naopak. Uživatel po dohrání dostal nějaké bodové hodnocení, podle kterého nebyl schopen určit co bylo špatně a co bylo dobře. Ladění v aplikaci mi přišlo velmi nepřesné.

1.2.2 Yousician

Yousician vyvíjí mobilní aplikace pro výuku hry na kytaru, baskytaru, ukulele, klavír a zpěv. Aplikace obsahuje hlavně výukové kurzy, které jsou kvalitně zpravované pomocí videí. Aplikace také obsahovala praktické hraní, kde uživatel musel hrát podle ukazatelů na strunách s čísly. Po dohrání skladby uživatel obdrží skóre, které následně může porovnat s ostatními uživateli v žebříčku. Žebříček bylo možné filtrovat. Uživatel si také mohl zvolit jeden z několika stylů hudby, který dával uživateli možnost zvolit si styl, který mu vyhovuje. Celkově aplikace byla obsahově bohatá.[3]

Aplikaci lze v omezeném obsahu používat bezplatně. Pokud by uživatel chtěl používat více nástrojů a všechny kurzy, tak musí platit měsíční předplatné.

Aplikace mi přišla zábavnější a měla modernější vzhled Lekce byli záživnější, ale možná o trochu méně informativní. V placené verzi si myslím, že by uživatele dokázala aktivně motivovat ke hraní.

1.2.3 Simply

Aplikace od Simply Piano fungovala podobně jako Yousician. Jednotlivé nástroje na rozdíl od Yousician byly rozděleny do jednotlivých aplikací. Simply obsahovala aplikace pro piáno, kytaru a zpěv. Jednotlivé lekce byly propojené rovnou s hraním na hudební nástroj. Aplikace v bezplatné verzi nabízela pouze minimum funkcí. Na rozdíl od Yousician, zde nebylo moc přehledné, co se uživatelovi odemkne v placené verzi, jelikož některé části aplikace byli zamčené celkově.[4]

Jedním bezplatným prvkem byla mini hra, ve které uživatel mohl cvičit rytmus a čtení not i bez potřeby mít u sebe hudební nástroj. C tomto případě mi přijde, že se jedná o dobrý doplněk, přestože to úplně nerozvíjí hraní na samotný nástroj, ale dodává možnost se rozvíjet v hudebních dovednostech i v případě, kdy nemá uživatel možnost používat samotný nástroj. V případě klavíru, který není snadno přenositelný, je tato možnost ještě více přínosná.

1.2.4 Practice Partner

Aplikace je zaměřená na procvičování skladeb s nahrávkou. Velkou výhodou byla možnost si nechat danou skladbu přehrát. V aplikaci byly placené jednotlivé skladby a pár zkušebních zdarma. Uživatel si mohl aplikaci zdarma vyzkoušet a zjistit, zda mu vyhovuje nebo ne.[5]

Bohužel zde nebyly noty k daným skladbám. Hráč na hudební nástroj si musel sám noty sehnat jinými způsoby nebo je musel odposlouchat z nahrávek. Pak mohl hrát s nahrávkou.

Jedná se o velmi jednoduché přesto efektivní řešení, které donutí hráče poslouchat okolí, které se hodí například při hraní ve skupině. A při možnosti hrát více skladeb by i uživatele mohlo motivovat ke hraní.

1.2.5 Perfect Ear: Music & Rhythm

Aplikace cílí na čtení z not, rozlišování hudebních intervalů, rozlišování molových a durových stupnic. Obsahuje i základní teorii k daným tématům. Aplikaci lze nastavit pro klavír, kytaru a housle. Toto nastavení hudebního nástroje, však aplikaci moc neupravilo, jen se na některých místech objevily hmatníky s aktuálně hranými (poslouchanými) notami, které nebyly příliš přehledné a použitelné. Jednotlivá cvičení mohl dělat kdokoli i bez znalosti těchto nástrojů. Aplikace obsahovala denní statistiky. UI aplikace bylo příjemné a jednoduché. Většina funkcí aplikace byla zdarma.[6]

V aplikaci bylo více funkčních chyb (občas aplikace nereagovala na některá tlačítka v aplikaci), ale koncept byl celkem efektivní a motivoval uživatele trénovat sluch, čtení z not, rytmus i další příbuzné dovednosti, které jsou spojeny s hraním na hudební nástroj nehledě, na který uživatel hraje. Dal se zde nastavit i denní plán trénování. Nebylo zde ale možné nastavit výchozí frekvenci ladění. Nastavení některých věcí bylo zbytečně ve společném hlavním menu. Toto menu bylo kvůli tomu velmi složité pro běžného uživatele.

1.2.6 MyEarTraining

Aplikace MyEarTraining je velmi podobná aplikace Perfect Ear: Music & Rhythm. Navíc zde obsahovala češtinu s občasnými chybami. Design aplikace byl zastaralejší a osobně mi přišel nezajímavý. Avšak většina věcí se dala nastavit přímo tam, kde to uživatel potřeboval. Na

rozdíl od Perfect Ear: Music & Rhythm aplikace obsahovala více bonusového obsahu, který byl zpoplatněn.[7]

1.2.7 Noutee

Mobilní aplikace obsahovala (s aplikací na stolní počítač) na procvičování látky, která je běžně probírána na hodinách hudební nauky v ZUŠ. Zde byl na rozdíl od předchozích aplikací velký výběr různých materiálů jak skladby tak třeba i popsané různé hry s žákem. Také na jejich stránkách bylo dostupné video s instrukcemi, jak je možné ve výuce danou aplikaci používat. Aplikace byla zaměřena pro mladší publikum jak svým designem, tak tématy. Všechna cvičení ale vycházela z těchto materiálů, takže se je člověk po nějaké době naučil podle toho jak to bylo minule. [8]

Aplikace nabízí možnost nahrání not z externích programů pomocí MusicXML. Jednotlivé funkce aplikace mi však přišly zbytečně složité na pochopení. Z vizuální stránky mi aplikace nepřišla moc zajímavá.

Licence pro použití aplikace je rozdělena do více úrovní. Některé funkce jsou dostupné bezplatně v časově limitované verzi aplikace, některé zpoplatněné levnější variantou licence a některé jsou pouze v dražší licenci. Licence je placena v ročních intervalech. Je zde možnost zakoupit Licenci pro více uživatelů najednou. Tyto nabídky jsou především mířené na školy. Škola pak rozdává aplikaci svým žákům. Tato školní licence může zpřístupnit aplikaci i studentům, kteří si ji sami nedokáži platit.

1.2.8 Sluchohry.cz

Webová aplikace obsahující kreativní hry spojené s hudební výukou, které nebyly v žádných jiných aplikacích. Jednotlivé hry byly celkem zábavné a dokázal bych si je představit hrát mladší publikum. Bohužel byly pouze čtyři. Aplikace byla plně v češtině a byla celá bezplatná. [9]

Jedna z her byla pojmenovaná „Posuvníky“, ve které uživatel musel podle poslechu složit část skladby pomocí posuvníků (drag-and-drop elementů), které ve skutečnosti představovaly noty, ale touto abstrakcí za posuvníky uživatel nepracoval přímo s notami. Druhou hrou bylo pexeso, které bylo složeno ze známých skladeb. Uživatel musel najít stejné skladby. Třetí hrou bylo krokování not, kde uživatel musel zapsat názvy not vždy posunuté o určitý interval znázorněný počtem šipek od poslední noty. Čtvrtou o trochu méně zajímavou bylo skládání skladby z částí. Zde často hrály skladby, které člověk neměl na-poslouchané v celém znění a neměl z čeho při skládání vycházet.

1.2.9 Výsledek zkoumání

Existující aplikace často necílí na samotné hraní na nástroj, a když už ano, není většinou kvalitně zpracované. Avšak mezi učiteli hudby je cíl přimět studenty cvičit i mimo výuku brán jako základní, na který by se tyto aplikace měli zaměřovat. Také způsob zpoplatnění často v těchto aplikacích není mířen na mladé studenty, kteří často nemají možnost použít aplikace v plném rozsahu, kde osekané verze, až na výjimky, nenabízí téměř nic. Na druhou stranu některé funkcionality aplikací mi přišly jednoduché, a přesto velmi dobře použitelné. Příkladem může být hraní s nahrávkou, kde pouze hraje nahrávka s metronomem.

Kapitola 2

Technologie spojené s vývojem na platformě Android

V této kapitole bude zaměřeno na technologie, které budou použity v praktické části aplikace.

2.1 Nástroje pro vývoj aplikací na platformě Android

Zde bude zvolen jeden z hlavních nástrojů pro vytváření moderních mobilních aplikací na zařízeních s OS Android. Od volby tohoto nástroje se bude dále vyvíjet výběr následujících technologií.

2.1.1 Flutter

Flutter je moderní Multiplafromní framework pro vývoj aplikací na Mobilní zařízení Android a iOS a také pro webové aplikace. Je napsán v jazyce Dart. Využívá vlastních UI komponent, není tedy závislý na komponentách připravených na jednotlivých zařízeních, což je jeho hlavní výhoda, ale i velká nevýhoda. Největší výhodou se mi zdála úprava aplikace v reálném čase. Když programátor upravil část kódu, hned se projevila na zařízení. Tyto změny totiž byly poslány na zařízení hned pomocí síťové komunikace. Pro složitější a propracovanější mobilní aplikace, ale není moc vhodný, protože používá vlastních UI elementů, a tak nové technologie napsané pro čistě Android nebo iOS nejsou dostupné. Člověk si je musí buď napsat sám, nebo počkat, až na to někdo napíše knihovnu, která bude dělat něco podobného.[10]

2.1.2 Nativní Kotlin

Nativní kotlin je nejrobustnějším řešením je v něm hned vše přístupné při přidání nové funkcionality na platformu Android, jelikož je nativně podporován Androidem. Hlavní nevýhodou je platformní závislost pouze pro zařízení s operačními systémy Android. Je globálně podporován a má velkou uživatelskou základku — velká řada problémů je již vyřešena někým jiným a lze používat také existující Java knihovny, kterých je mnoho.[11]

2.1.3 Kotlin Multiplatform mobile (KMM)

KMM je nová technologie pro vývoj multiplatformních aplikací jak pro Android, tak iOS. Na rozdíl od technologie *Flutter* nevytváří vlastní komponenty, pouze pracuje jako společný platformní backend mobilních aplikací. Samotné UI se musí na každé platformě vytvořit samostatně. Lidé často u KMM nachází problém při překladu Kotlinu do *Objective-C* pro iOS. Komunita KMM není velká. Přechod z Android aplikace na KMM není složitý ,jelikož používá stejný jazyk a většinu knihoven. Pro iOS je Kotlin přeložen do *Objective-C*. [12], [10]

2.1.4 Výběr technologie

Pro tuto aplikaci jsem zvolil nativní Kotlin hlavně kvůli velké uživatelské základně. Nativnímu podporování všech služeb na platformě Android. jelikož se nejedná o jednoduchou zobrazovací aplikaci. Dále také pro jednoduchý přechod platformního backendu aplikace na KMM, kde by se v budoucnu mohla vytvořit podobná aplikace pro zařízení iOS, ve které by společná logika byla zachována v jediném platformním backendu.

2.2 Android Studio

Android Studio je oficiální integrované vývojové prostředí (IDE) pro vytváření aplikací pro Android, které vyvíjí a spravuje společnost Google spolu s JetBrains. Jednou z hlavních výhod používání IDE Android Studio je bohatá sada vývojových nástrojů, které zjednodušují proces vytváření, testování a ladění aplikací pro systém Android. Mezi tyto nástroje patří:

Editor kódu umožňuje snadnou práci s projektem a zdrojovými kódy pomocí automatického doplňování, barvení klíčových slov, označování nového kódu v daném *git* repozitáři a mnoho dalších.

Integrované sestavení a instalace je nástroj, který umožní sestavit aplikaci a následně instalovat na vybrané zařízení bez nutnosti dalších akcí.

Emulátor umožňuje vytvořit mnoho různých virtuálních Android zařízení, na kterých je možno testovat aplikaci.

Inspekce překreslení je nástroj pro sledování, kde v Compose UI 2.4.2 dochází k překreslení obrazovek. Na základě toho lze optimalizovat toto vykreslování v případě výkonnostních problémů.

Průvodce úložiště na zařízení umožňuje přístup k datům přímo v zařízení, kopírováni, mazání a přesouvání jednotlivých souborů.

Krokování programu umožňuje programátorovi zastavit běh programu v téměř jakémkoli bodě a zjistit, co se přesně v dané části kódu děje.

Lint je nástroj pro základní statickou analýzu kódu. Tuto analýzu provádí v reálném čase, tedy není třeba projekt sestavit. Je ale pouze zobrazena nad několika posledními otevřenými soubory. Pro kompletní analýzu je možno spustit samotný test.

Plugins jsou rozšíření často třetích stran, které dále zjednodušují práci v prostředí. Mezi tyto rozšíření patří i aktuálně velmi často zmiňované *GitHub Copilot*, který pomocí umělé inteligence dokáže předpovědět a doplnit části kódu, které se často opakují. Na složitější programování je lepší ho raději vypnout jelikož vytváří spoustu chyb (v aktuální verzi).[13]

A mnoho dalších ...

Tato množství nástrojů některým nevyhovuje, jelikož to zatěžuje zařízení, na kterém je vyvíjeno, což vede k neplynulému běhu IDE.

2.3 Kotlin

Aplikace pro Android se píší převážně v jazyce Kotlin a Java. Kotlin se připojil jako hlavní podporovaný jazyk k jazyku Java v roce 2017.[14] Silnou stránkou Kotlinu je zpětná kompatibilita s Javou, která umožňuje využít existující Java řešení a knihoven v Kotlin projektu nebo rozšířit existující Java aplikaci pomocí Kotlinu. Kotlin propojuje výhody objektově orientovaného programování s funkcionálním programováním.

2.3.1 Robustnější syntaxe

Syntaxe kotlinu se snaží být minimalistická a zároveň velmi čtivá, což vede k menší chybovosti a lepší čitelnosti výsledného kódu. Mezi hlavní syntaktické výhody patří:

Nullable operátory a typy bez možnosti zapsání null hodnoty [15].

Extension funkce rozšiřující funkce nad již existujícími objekty, které můžou být jak privátní v daném kontextu, tak globální [16].

Delegation vytvoření objektu či proměnné při běhu aplikace až ve chvíli, kdy je daný objekt / proměnná potřeba [17].

Immutability možnost využití interních neměnitelných (*immutable*) objektů a proměnných označených klíčovým slovem val, které zamezí úpravám těchto hodnot [18], .

Type Inference k automatickému určení typu proměnné nebo výrazu se zachováním vlastností staticky typovaného jazyka[19]. [20]

2.3.2 Coroutines

Pro asynchronní neblokující práci se v Kotlinu používají *Couroutines*, které fungují jako takzvané „light weight“ vlákna, u kterých se na rozdíl od běžných vláken nedefinuje přesně, co se má spustit na novém vlákně, ale definuje se množina vláken a kód, který se na tyto vlákna má rozdělit. To umožňuje spustit i několik tisíc vláken najednou a aplikace poběží bez problému dál. Samotné rozdělení na vlákna řeší při běhu aplikace samotný Kotlin. Množina vláken, na kterých se dané operace mají provádět, je uložena v *CoroutineContext*, který obsahuje řadu informací o *Coroutine*, mimo jiné i *CoroutineDispatcher* třídu, ve které je definováno, jaká vlákna se pro danou *Coroutine* mají použít. [21]

Podle Android Developers [22] jsou v programátorském rozhraní (API) androidu 3 připravené Dispatcher objekty:

Dispatchers.Main obsahuje hlavní vlákno, na kterém se provádí operace spojené s uživatelským rozhraním (UI) a jeho logikou. Je možné zde provádět i výpočetně nenáročné operace, které nebudou mít vliv na rychlost vykreslování jednotlivých obrazovek.

Dispatchers.IO obsahuje neprázdnou množinu vláken, která je určená (a optimalizovaná) pro práci s vstupně výstupními daty (IO). V této práci je vhodná například pro čtení audio dat z mikrofonu. Dále je také využívána na síťové komunikace a čtení dat z disku zařízení.

Dispatchers.Default obsahuje neprázdnou množinu vláken, která se používají na výpočetně náročné operace, které nejsou spojeny ani s UI, ani práci s IO daty. V tomto projektu je použit například pro transformaci audio dat pomocí DFT 3.2.1.

Kotlin z *suspend* funkcí (*suspend* funkce se dají pouštět pouze z *Coroutines*) vytvoří třídu implementující rozhraní *Continutation*. Při běhu aplikace sám určí a optimalizuje, kdy a jak dojde k samotnému přepínání práce mezi jednotlivými *suspend* funkcemi. Tímto způsobem (práce

s vlákny a asynchronním programem) se výrazně sníží režie spojená s přepínání kontextů a voláním do operačního systému při přepínání vláken. Podobný přístup můžeme nalézt také v programovacím jazyce Go. [20]

2.3.2.1 Komunikace mezi Coroutines

Suspend funkce mohou vracet hodnoty, ale často je potřeba vytvořit neblokující kontinuální čtení. V této práci se jedná o čtení dat z mikrofonu. Pro návrat více hodnot lze použít List, ale pomocí něho nejsme schopni vracet data kontinuálně pouze jednorázově seznam hodnot po konečném zpracování všech dat. Složitá práce s vlákny, zámky, uspáváním a notifikací vláken se v Kotlinu a *Coroutine* nepoužívá. Pro komunikaci mezi různými *Coroutine* se dnes v Kotlinu používají především *Flows*, které jsou nadstavbou nad dříve používanými *Channels*, a k většině problémů není třeba používat *Channels* samotné. Existuje velká řada různých *Flows*, které jsou připraveny pro často se opakující situace v programování spojené s prací s asynchronním programováním a *Coroutines*. *Channels* lze využít pokud neexistuje připravená *Flow* pro aktuální případ a získat tím výkonnostní výhodu oproti neoptimálnímu *Flow* řešení. Avšak musíme počítat se zhoršenou čitelností rozšiřitelností a udržitelností kódu.[23], [24]

API pro *Flows* je připravené přímo pro konkrétní problém, takže je intuitivní. *Flows* také řeší uzavřenost na zapisování. Podobně jako u kolekcí je zde *MutableFlow*, do které je možné zapisovat nové hodnoty, a *Flow*, ze které lze pouze data číst. Existují zde i operace pro modifikaci dat ve *Flow* jako je třeba známá funkce *map* nebo *filter*. Lze i různé *Flows* spojovat a kombinovat. [23], [24], [25]

V Androidu se *Flows* používají prakticky na vše co obnáší potřebu znát změnu hodnoty v reálném čase. Téměř vždy se pak vyskytnou při přenosu dat z *ViewModel* do *Compose UI* 2.4.2, kde se pro tyto účely používá StateFlow. (Pro přenos do obrazovek definovanými XML se používá jiná struktura LiveData). Tato *Flow* vždy definuje stav, který se má zobrazit na samotném UI. To umožňuje aktualizace obrazovek v reálném čase, kde se tyto Flows přepíšou do *Composable* stavů a při změně hodnoty *Flow* se také změní hodnota Stavu a část obrazovky se aktualizuje. [25]

2.4 Android

Tato část se zaměří specificky na technologie spojené s nativním vývojem na zařízeních - Android při použití nativního vývoje pomocí programovacího jazyka Kotlin.

2.4.1 Buffers

Buffers v Androidu jsou speciální konstrukce pro ukládání dat, do kterých se zapisuje nebo čte lineárně (od začátku do konce). *Buffers* obsahují pozici, od které se mají další data zapisovat, alokovanou kapacitu, maximální dovolenou kapacitu, do které se má zapisovat, která je maximálně velká jako maximální kapacita nebo menší. Tyto *Buffers* jsou optimalizované na zapisování a čtení dat kontinuálně jedním směrem. Používají se při čtení dat z disku a z různých čidel (při IO operacích). [26]

2.4.2 Compose

Pro tvorbu uživatelského rozhraní (UI) aplikace byla využita moderní UI knihovna *Compose (Compose UI / Jetpack Compose)*, která postupně začala nahrazovat předchozí postup pomocí XML obrazovek. V *Compose* je možno vytvářet jednotlivé části obrazovky i s logikou (například animacemi). Jednotlivé komponenty obrazovky jsou skládány pomocí Kotlin funkcí označených

anotací *@Composable*. Funkce jsou součástí kotlin souborů, ale nejsou součástí explicitně definovaných třídy. Jednotlivé části jsou postupně vnořovány do sebe a celek tvoří stromovou strukturu.

Tyto *Composable* části je možno nezávisle a snadno znovu použít v konkrétním projektu i v jiných projektech. Tento fakt umožňuje snadno vytvářet vlastní knihovny s různými částí uživatelského rozhraní, které lze používat napříč aplikacemi. Komponenty se také dají snadno přesouvat na jiná místa obrazovky.

Compose se aplikuje uvnitř *Activity* ve lifecycle funkci *onCreate* funkcí *setContent*, kde v jejím parametru *content* se už pracujeme v *Compose* prostředí, kde se používají *Composable* funkce. Pokud celá aplikace využívá *Compose UI*, tak toto nastavení je v hlavní *Activity* třídě projektu.

Z tohoto nastavení *Compose* uvnitř *Aktivity* je zřejmé, že *Compose* obsahuje původní životní cyklus *Aktivity* stejně jako předchozí způsob pomocí *Aktivit* a XML obrazovek. *Compose* navíc ještě obsahuje vlastní životní (kompoziční) cyklus. Compose překresluje *Composable* funkce (*Recomposition*) při změně hodnoty parametru funkce nebo při změně hodnoty stavové instance uvnitř funkce. Při překreslení části se často překreslí i všechny její pod-části. Tento proces překreslování je stále ve vývoji a není zatím přesně definován. Vývojáři *Compose* se snaží minimalizovat překreslení jednotlivých komponent co nejvíce a tuto záruku překreslení všech pod-částí nedávají. [27]

Časté překreslování velkých částí obrazovky je výpočetně náročné a obrazovka se může vykreslovat pomalu nebo animace na obrazovce nebudou plynulé. Pokud nechceme překreslit danou část obrazovky, ale pouze její pod-část, předáváme hodnotu stavové proměnné části, která nemá být překreslena parametrem obsahující referenci na funkci, která se při změně hodnoty nezmění, ale při zavolání funkce vrací hodnotu, která při změně překreslí pod-část. V parametru pod-části která se má překreslit zavoláme tuto funkci.[27], [28] Viz Výpis kódu 2.1

Listing 2.1 Vyhnutí se zbytečnému překreslení

```
@Composable
fun ExampleScreen(viewModel by viewModel()){
    // State property delegate
    val textToShow by viewModel.text.collectAsStateWithLifecycle()
    // Lambda function that returns textToShow
    ExampleFragemnt(textToShow = {textToShow})
}

@Composable
private fun ExampleFragment(textToShow: () -> String) {
    Text(textToShow()) // Call to lambda to get value of textToShow
}
```

V tomto případě se při změně textu *textToShow* překreslí pouze část s textem (funkce *Text*). Pokud by byl předán parametr běžným způsobem pomocí parametru obsahující textový řetězec (*String* parametru), došlo by k překreslení *ExampleScreen* a všech jeho pod-částí obsahující daný parametr, přestože *Composable* funkce parametr nepoužívá, pouze ho předává svým pod-částem. Časté překreslování velkých částí obrazovky je výpočetně náročné. [28]

Pro kontrolu a testování spojenou s překreslováním lze využít nástroje přímo v *Android Studiu* s názvem *Layout Inspector* 2.2

2.4.2.1 Canvas

Pro velmi specifické případy se dá vykreslit vlastní komponenta za použití komponenty *Canvas*, která dovoluje nativní vykreslování přímo v pixelech. V prostředí *Canvas* je možné vykreslit nativně různé základní tvary přímky a také bitmapové obrázky. Pro tyto účely jsou

v *Canvas* připravené funkce. Je potřeba ale dbát na to, že v tomto nativním vykreslování se již nepracuje s display *pixels* (DP), které definují optimální velikosti minimálně velkých bodů na konkrétním zařízení a jsou hlavní jednotkou při práci s *Composable* funkcemi, ale s běžnými *pixels* (PX). Převedení DP na PX je možné pomocí funkce *toPx* na instanci třídy DP. Převod z pixelů na DP je složitější a je k němu potřeba využít *LocalDensity* třídu, která je definovaná v kontextu *Composable* funkcí.[29], [30]

2.4.2.2 Navigace

Pro navigaci mezi různými obrazovkami v *Compose UI* frameworku se používá *Composable* funkce *NavHost*, ve které jsou definovány všechny cílové destinace. Tento *NavHost* funguje jako přepínač potomků ve vykreslovacím stromu. *Navigation Controller*, který je předán *NavHost* funkci, obsahuje zásobník s navigačními cíli, které byli již navštíveny. Pomocí tohoto zásobníku se provádí samotná navigace. [31]

Cílové *Composable* části jsou v *NavHost* funkci definovány ve funkci *NavHost.composable()*, pomocí URI řetězce, který může obsahovat i parametry, a počáteční destinaci navigace. [31] Příklad definice navigace: 2.2.

Listing 2.2 Navigace v Compose

```
@Composable
fun MyNavigation(){
    val navController = rememberNavConttroller()
    NavHost(navController = navController, startingDestination = "seznam"){
        composable("seznam"){
            SeznamScreen(navigateToDetail = {id ->
                navController.navigate("detail/$id")
            })
        }
        composable("detail/{id}") { backStackEntry ->
            DetailScreen(backStackEntry.arguments?.getString("userId"))
        }
    }
}
```

Z kódu je vidět, že práce s těmito URI je celkem komplikovaná a náchylná na chybu při špatném zapsání URI řetězce. Také neřeší typ, který z daného URI parametru obdržíme. Komplexnost roste s rostoucím počtem navigovaných míst a počtem parametrů *Composable* funkcí.

2.4.2.2.1 Compose Destinations

Tyto nedostatky řeší knihovna *Compose Destinations*, která pomocí Anotací a generovaného kódu vytvoří tuto navigaci tak, že nedovolí vytvořit nesmyslnou URI a navigace i s parametry bude bezpečně typovaná. Stačí přidat k *Composable* funkcím anotaci *@Destination*. Následné navigování pomocí funkce *NavController.navigate()* neobsahuje parametr s URI řetězcem, ale vygenerovanou Třídu knihovnou s názvem odpovídající názvu *Composable* funkce a příponou *Destination* tato třída má konstruktor odpovídající parametrům *Composable* funkce, který zajistí typovou bezpečnost chybějící v URI parametrech. [32]

Vygenerovaný kód lze následně použít samostatně nebo přidat do originální *Compose* navigace. Díky tomu je možno zajistit kompatibilitu i s běžnou navigací i s dalšími rozšiřujícími knihovnami pro navigaci jako je například *AnimatedNavHost* pro animované přechody v navigaci od *Accompanist* knihovny[33]. [32]

2.4.2.3 Material Design

Nedílným rozšířením knihovny *Jetpack Compose*, je knihovna *Material3* od *Matereial Design*. Knihovna připravuje základní komponenty, kterými jsou tlačítka, listy, texty, vyskakovací okna a mnoho dalších základních komponent, které mezi sebou graficky tvoří jeden celek. Musí se zde, ale správně nastavit paleta barev, a také správně pracovat s výškovými vrstvami.[34]

Vlastní komponenty je možno skládat z existujících komponent v *Material Design* nebo ze základních komponent obsažených v *Jetpack Compose* knihovně samotné. Nové komponenty v *Material3* knihovně lze kombinovat s předchozími v *Material2* knihovně, ale programátor musí myslet na to, že nastavení barev nebude kompatibilní, a tak musí *Material2* prvky přebarvit ručně. Android Delelopers toto použití *Material2* komponent s *Material3* komponentami nedoporučuje, pokud to není nutné (například komponenta zatím *BottomSheetScaffold* není vytvořena pro *Material3*) [35]

2.4.3 Dynamické barvy

Na Androidu 12 lze nastavit v aplikaci dynamické barvy, které se přizpůsobují barvám zvoleným na tapetě zařízení. Tato volba barev cílí na přizpůsobení aplikace zvolenému uživateli. Barvy lze nastavit jak pro *Compose UI*, tak pro XML obrazovek. Tento přístup umožňuje personalizovanější a přizpůsobivější uživatelský zážitek, protože umožňuje rozhraní plynule splynout se vkusem uživatele. Barvy jsou hlavně voleny z domovské obrazovky zařízení a dále částečně nastavitelné v nastavení telefonu (záleží na konkrétním zařízení).[36]

Při testování se však ukázalo, že použití dynamických barev může být pro uživatele matoucí. Pokud si uživatel nastaví pozadí červené, pak rozdíl mezi hlavní barvou tématu a chybovou barvou je minimální, což může být pro uživatele matoucí.

2.4.4 Oprávnění

Android podobě i iOS vyžaduje oprávnění (permission) pro některé operace pro zabezpečení svých uživatelů. V Android API existují dvě úrovně oprávnění. oprávnění při instalaci a oprávnění za běhu aplikace, kde oprávnění při instalaci (*Install permissions*) jsou vyžadována pouze v konfiguračním souboru projektu. Oprávnění za běhu aplikace (*Runtime permission*) vyžadují uživatelský souhlas při běhu aplikace. Oprávnění, která by mohla výrazně narušit soukromí a bezpečnost uživatele, je nutno vyžádat při běhu aplikace. Příkladem je nahrávání zvuku nebo videa. Oprávnění provádět tyto operace je nutno získat uživatelem až při běhu aplikace.[37]

Pro práci s *Runtime permissions* je v Compose UI připravena knihovna od *Accompanist*, která zjistí, zda dané oprávnění je již povoleno, nebo potřebuje udělení souhlasu. Pokud zjistí, že k danému oprávnění nemá přidělen souhlas a není blokováno operačním systémem (oprávnění jsou blokována operačním systémem při opětovném udělení nesouhlasu), otevře systémové okno, kde uživatel získá možnost udělit, nebo neudělit souhlas s používáním tohoto oprávnění. Aktuální stav oprávnění je možné zjistit přímo ve stavu, který je předán knihovnou.[38]

2.4.5 Persistence

Persistence systému Android označuje proces ukládání a správy dat v aplikacích systému Android, který programátorům umožňuje přístup k datům a interakci s nimi. Dvě běžné metody implementace persistence dat v aplikacích pro Android jsou *SharedPreferences* a *SQLite*.

SharedPreferences je mechanismus ukládání dat, který vývojářům umožňuje ukládat malé množství dat, typicky dvojice klíč-hodnota, do strukturovaného souboru XML. Tato metoda je nejvhodnější pro ukládání jednoduchých typů dat, jako jsou uživatelské preference nebo nastavení aplikace, které nevyžadují složité dotazování nebo manipulaci. K *SharedPreferences*

se běžně přistupuje pomocí *DataStore* knihovny, která předává hodnotu uloženou v *SharedPreferences* pomocí *Flow*, která signalizuje aktualizaci dat v reálném čase. *DataStore* také umožňuje definovat vlastní styl ukládání, kde je možné uložit složitější struktury (například objekty) pomocí vlastní definice. Například ve formátu JSON. [39]

SQLite je samostatný, bez-serverový a transakční databázový systém SQL, který vývojářům umožňuje spravovat složitější datové struktury. Systém Android obsahuje vestavěnou podporu *SQLite*, která aplikacím umožňuje snadno vytvářet, dotazovat a aktualizovat místní databáze. *SQLite* je ideální pro situace, kdy vývojáři potřebují ukládat a spravovat větší datové soubory, provádět složité dotazy nebo vytvářet vztahy mezi datovými entitami. Pro práci s *SQLite* se doporučuje knihovna *Room*.[40]

2.5 Architektura

Android Developers doporučuje rozdělit aplikaci do tří vrstev: vrstvu s UI, vrstvu s Daty, a volitelnou vrstvu pro modifikaci dat. Data by měla používat vždy jediný zdroj pravdy — vždy by měla v aplikaci být pouze jedna primární cesta, kde získat věrohodná data. Doporučuje také použití DI 2.5.1, pro menší provázanost mezi jednotlivými částmi aplikace. Vrstva s UI musí být dále rozdělena na vrstvu obsahující prezentovaná data a vrstvu s samotnými vykreslenými UI elementy.[41] Tato spolupráce na UI vrstvě je blíže popsána v 2.5.2.

2.5.1 Dependency Injection

Dependency Injection (DI) je technika používaná ke zjednodušení práci s architekturou aplikace, kde je definován postup vytváření tříd odděleně od samotného použití jednotlivých tříd. A v samotných třídách, kde jsou závislé třídy potřeba přidělit z DI, se tyto třídy aplikují v konstruktorech. Do těchto konstruktorů se předávají většinou z globální definice (tato definice nemusí být globální důležité že je oddělená). To zmenší provázanost kódu a zjednoduší náhradu jednotlivých tříd za jiné a to i pro testování, kde je možno nahradit tyto třídy za třídy testovací.[42]

Pro DI byla zvolena knihovna Koin, přestože hlavní podporovanou knihovnou pro DI na Androidu je *Hilt*. *Hilt* a *Koin* jsou velmi rozdílné při plnění DI. *Hilt* vytváří DI při kompilaci pomocí generovaného kódu z Java anotací. *Koin* funguje na základě *lazy* delegací z kotlinu při běhu aplikace. *Hilt* přináší malou výhodu v rychlosti, širší možností exotických použití DI frameworku, kompatibilitu s Javou. Koin na druhou stranu nabízí snazší implementaci, o něco kratší kompilaci a hlavně kompatibilitu s KMM 2.1.3. [43], [44]

2.5.2 Architektura vrstvy uživatelského rozhraní

Pro DI vrstvu na Android platformě je potřeba pracovat se stavem obrazovky a se samotným vykreslením odděleně. Jelikož například při rotaci obrazovky se celá obrazovka na zařízeních Android sestaví od začátku všechna data jsou při této operaci ztracena, proto je potřeba uchovávat informace o obrazovce v jiné vrstvě. V aplikaci následně samotnou grafiku představují *Views* a uchovávaný stav obrazovky je uložen ve *ViewModel* třídě. [45] Pro spolupráci těchto dvou vrstev se používají hlavně tří základní postupy:

MVVM View jsou prezentována data, která se mění v průběhu času a na View je třeba reagovat na tyto změny. Pro každá data je vytvořen vlastní datový tok. Komunikace opačným směrem, tedy z *View* k *ViewModels*, je pomocí funkcí s akcemi, kde je opět těchto funkcí několik (pro každou jednu operaci by měla být definována nová funkce).

MVI funguje podobně jako MVVM, ale místo definovaných několika funkcí pro komunikaci z *View* k *ViewModel* definuje jeden globální datový tok, do kterého jsou posílány všechny akce

(*Events*), a na *ViewModel* se následně rozdělí na jednotlivé funkcionality. Podobně i ve směru od *ViewModel* k *View* je definován jeden globální stav, který obsahuje všechny proměnné. Tedy celá obrazovka je definována jedním stavem.

MVP *MVP* propojuje vrstvu *View* s *ViewModel* provázaně. Tento postup není použitelný v případě použití *Compose UI* knihovny.[46]

Pro tuto aplikace se nejvíce vyhovuje architektura MVVM. Práce s více stavy, které definují stav obrazovky je náročnější na údržbu a rozšiřitelnost, ale hlavní výhodou oproti MVI je, že není třeba překreslovat při změně jednoho stavu celou obrazovku ale pouze jednotlivé části 2.4.2. V MVI se vždy změní jeden globální stav a to překreslí všechny části, které potřebují poslouchat na změnu stavu, nehledě na to, zda tato změna stavu definuje aktuální *Composable* část.

2.5.3 Moduly

Projekty je možné rozdělovat do více modulů. Tyto moduly rozdělují projekt do více částí, které zajistí lepší rozšiřitelnost, testovatelnost, rychlejší kompilaci (moduly, které nebyly změněny se nekompilují znovu), uzavřenost kódu na menší části. Nevýhodou je však větší komplexita projektů, která u menších projektů může být zbytečná.[47]

V tomto projektu je vhodné rozdělit projekt na alespoň 2 moduly. První modul pro práci s daty a některé UI prvky, které lze snadno použít ve stejném formátu i v dalších Aplikacích. Při tvorbě další podobné aplikace není třeba vytvářet a udržovat dvakrát stejnou logiku. Druhý pro aplikačně specifický kód. Bližší popis modulů níže:

common Společné znovu použitelné části, které bude možné využít v aplikacích, které nebudu specificky navrženy pro housle. Společná část nesmí být závislá na části aplikačně specifické. Modul nebude fungovat jako android aplikace ale bude v režimu android knihovny. Nebudou se zde definovat části spojené se samostatnou android aplikací. (Aplikační ikona, styly aplikace, název aplikace…)

violin Aplikačně specifická část, která bude obsahovat části specifické pro aplikaci na housle. (Jednotlivé obrazovky spojené pouze s aplikací zaměřenou na housle, definice jednotlivých strun…)

Závislost mezi moduly je definována v sestavovacím skriptu projektu 2.6 v části *dependencies*. Tato závislost by nikdy neměla být oboustranná. Oboustranná závislost znemožňuje nezávislé použití modulu v jiném projektu. Toto rozdělení do modulů pak nedává žádný smysl pouze zvyšuje komplexnost řešení. V tomto projektu bude modul violin závislý na common. [47]

2.6 Sestavení projektu

Pro sestavení Android projektu se běžně používají *gradle* skripty. Dnes se používají hlavně dvě verze *gradle* skriptů *Groovy* a *Kotlin*, Kde *Groovy* je starší a *Kotlin* novější verzí. Vesměs fungují stejně, jen mají trochu odlišnou syntaxi, oba tvoří takzvanou *Domain Specific Language* (DSL) strukturu pro nastavení a sestavení aplikace.[48]

V těchto souborech se nastavují parametry pro správné sestavení projektu. Programátor si musí dát pozor na to, že to nejsou konfigurační soubory, ze kterých se přečtou všechny parametry a podle těch se program sestaví. Projekt se sestaví pomocí tohoto skriptu, ve kterém záleží na pořadí jednotlivých částí.

Nastavují se zde externí knihovny. *Plugins*, které se následně používají při sestavování projektu, jako jsou například speciální části skriptu pro sestavení android aplikace nebo *plugins* pro generování kódu různými knihovnami. Dále samotné nastavení Android projektu, jaký má použít kompilátor pro sestavení *Composable* funkcí, zda v projektu bude použit *Compose* nebo *View Binding* (knihovna, pro jednoduší práci s *XML layouts*), podporované verze Androidu a další…[48]

2.6.1 Bill Of Material

Některé větší knihovny složené z více dílčích knihoven používají *Bill Of Material* (BOM), který obsahuje verze k jednotlivým dílčím knihovnám, ze kterých je větší knihovna složena. Údržba těchto knihoven je díky BOM snazší. Uživatelé knihoven nemusí neustále sledovat, zda pro některou část knihovny vyšla nová verze. Uživatelům stačí pouze zkontrolovat aktuální verzi BOM. BOM používají například knihovny jako je *Compose* nebo *Firebase.*[49]

2.6.2 Verzovací katalogy

Pro jednodušší práci s gradle skripty nad více moduly je vhodné použít verzovací katalog, který definuje jednotlivé knihovny a pluginy odděleně od samotných sestavovacích skriptů, určených k sestavení projektu. Verzovací katalog se pak použije ve všech modulech stejný. Všechny knihovny jsou definovány pouze jednou ve verzovacím katalogu a nestane se, že v různých modulech budou různé verze té stejné knihovny, které nebudou mezi sebou kompatibilní. Také je zde možné vytvořit takzvané balíčky knihoven, pomocí kterých je možné vytvořit vlastní seznam několika knihoven a při nastavení použití balíčku se použijí všechny knihovny najednou. Toto je velmi užitečné například, při nastavování *Compose* knihoven, jelikož *Compose* se skládá z celé řady různých dílčích knihoven. Tyto knihovny se pak dají aplikovat najednou. [50]

Pro verzovací katalogy byl zvolen nástroj *Tom's Obvious Minimal Language* (TOML), který definuje jednotlivé verze knihovny, balíčky knihoven, a pluginy v jednom konfiguračním souboru. Tento soubor se následně nastaví do *gradle* skriptu, který z něj vytvoří verzovací katalog. Tento soubor je definován pouze jednou v hlavní části projektu a v jednotlivých modulech se vždy použije jeden. Dále nám tento katalog umožňuje vlastní pojmenování knihoven tak, že například ve více projektech budou používány různé knihovny pro DI a všechny tyto knihovny je možné zapsat s prefixem „di". Našeptávač IDE pak ukáže všechny knihovny, které jsou připravené pro DI, při zadání „di" v Android Studiu. [50], [51]

2.6.3 Nahrávání zvuku

Audio data pro další možnou transformaci a úpravu budou získána v *Pulse-Code Modulation* (PCM) formátu, který převede analogové zvukové vlnění do pravidelných digitálních záznamů, které odpovídají stavu analogového signálu v odpovídajícím čase.[52]

Android podporuje 8-mi bitové nebo 16-ti bitové kódování v PCM, které je pravidelně segmentované. Pro získání naměřených audio dat na zařízeních Android v PCM existuje API, které používá třídu *AudioRecord*, u které se musí nastavit parametr *audioFormat* na *PCM_16*. Tato třída bude periodicky zapisovat naměřená data do odpovídajících Buffer 2.4.1. Jelikož se bude zapisovat neustále, je vhodné použít *Coroutine* na definovaných vláknech pro zápis a čtení dat *Dispatcher.IO* 2.3.2. [53]

Kapitola 3

Popis tónu a not a zpracovaní zvukových dat do daných tónů

Tato kapitola popíše stručně problematiku tónů a not. Následně popíše postupy pro získání frekvence hrané na hudební nástroj pomocí mobilních zařízeních s OS Android.

3.1 Tóny a noty

Lidé vnímají výšku tónů logaritmicky. Jedna oktáva je dvojnásobná frekvence původního tónu dvě oktávy čtyř-násobek původní frekvence atd... Existuje několik různých členění tónů. Nejběžnějším v klasické hudbě je 12-ti tónové rovnoměrně temperované ladění (12-ET), ve kterém je 12 tónů v jedné oktávě. Následující tón je možno získat pomocí následujícího vzorečku: $2^{(1/12)}$ krát původní nota. Z frekvence jednoho tónu lze postupně dopočítat všechny ostatní tóny.[54]

Tyto tóny jsou následně zapisovány do notového zápisu a říká se jim noty. V notovém zápisu je v jedné oktávě 8 různých not. Zbylé 4 tóny do původních 12-ti v 12-ET jsou vytvořeny pomocí posuvek, které můžou být přiděleny k jednotlivým notám.

♭ čteno „bé“ posouvá tón odpovídající původní notě bez jiné posuvky o násobek $1/2^{(1/12)}$. (výsledná frekvence je tedy nižší)

♯ čteno „křížek“ posouvá původní tón odpovídající notě bez jiné posuvky o násobek $2^{(1/12)}$.

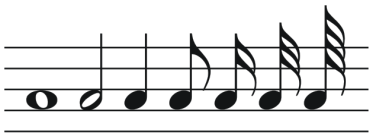
Tyto posuvky lze přiřadit kterékoli notě. Ale například nota F s posuvkou ♭ je ekvivalentní notě E bez posuvky, jelikož oba zápisy odpovídají stejnému tónu v 12-ET. Zápis F♭ se většinou nepoužívá, je spíše matoucí pro čtenáře notového zápisu.

Notový zápis dále obsahuje klíč, ve kterém jsou dané noty zapisovány. Různé hudební nástroje používají různé klíče pro snazší čitelnost not. Tyto klíče upravují frekvenční rozmezí, ve kterém je daný notový zápis. Některé nástroje hrají vysoko, jiné zas nízko (frekvenčně). Některé nástroje mohou používat i několik klíčů současně(např. Klavír).

Notový zápis obsahuje předznamenání, které určuje tóninu. Tato tónina určuje předurčené posuvky pro větší část nebo celý notový zápis. Tyto posuvky z předznamenáni lze vyrušit pomocí značky ♮ přiřazené konkrétní notě.

Jednotlivé noty v notovém zápisu také definují dobu, po kterou mají být hrány. Tyto noty pak mají různé grafické znázornění. Viz obrázek 3.1

Notový zápis obvykle obsahuje další řadu upravujících popisů, které nebudou v této aplikaci v aktuálním stavu vývoje příliš užitečné.



Obrázek 3.1 Různé grafické podoby různě dlouhých not. Grafika z wipedia.org [55]

3.2 Získání frekvence zvuku

Zvuk je Analogový signál (vlnění) šířící se vzduchem. Pro ladičku a zpětnou vazbu uživateli bude ze zvuku potřeba získat jeho frekvenci.

Před nahráváním zvuku je také zapotřebí získat oprávnění pro nahrávání zvuku *android.permission.RECORD_AUDIO* 2.4.4. Bez tohoto oprávnění se aplikace ukončí s chybou.

3.2.1 Diskrétní Fourierova Transformace

Pro získání frekvence ze zvukových dat je vhodná diskrétní Fourierova Transformace (DFT). Pomocí této transformace je možno převést audio data v časové doméně obsahující pravidelně segmentované audio záznamy do domény frekvenční.

Pro tuto transformaci jsou potřeba pravidelně segmentovaná audio data z původního analogového audio signálu. Tato data je možné získat na zařízeních Android ve formátu PCM 2.6.3.

Množství vstupních dat pro DFT musí být alespoň $2x$ kde x je maximální hledaná frekvence. Tedy pro frekvenci 2000Hz musíme mít alespoň 4000 vstupních záznamů a zařízení, které dokáže zaznamenat alespoň 4000 záznamů za sekundu.

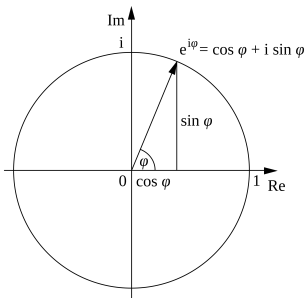
DFT spočívá ve vytvoření transformační matice (DFT matice) (viz níže), kde na k-tém řádku (indexováno od 0) jsou hodnoty, které odpovídají sinu a cosinu s frekvencí k pro daná data. DFT matice má tedy pro data z jedné sekundy na řádku 0 sinus i cosinus o frekvenci 0 Hz, na řádku 1 sinus a cosinus o frekvenci 1 Hz atd... Pro uložení hodnoty jak sinu, tak cosinu se používají obvykle komplexní čísla, kde v reálné části komplexního čísla jsou hodnoty cosinu a v imaginární hodnotě jsou hodnoty sinu. [56]

n = Počet naměřených audio hodnot

$$A_{n,n} = \begin{pmatrix} a_{0,0} & a_{0,1} & \cdots & a_{0,n-1} \\ a_{1,0} & a_{1,1} & \cdots & a_{1,n-1} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n-1,0} & a_{n-1,1} & \cdots & a_{n-1,n-1} \end{pmatrix} = \begin{pmatrix} A_0 = (a_{(0,0)}, a_{(0,1)}, a_{(0,2)}, \dots) \\ A_1 = (a_{(1,0)}, a_{(1,1)}, a_{(1,2)}, \dots) \\ A_2 = (a_{(2,0)}, a_{(2,1)}, a_{(2,2)}, \dots) \\ \vdots \\ A_{n-1} = (a_{(n,0)}, a_{(n,1)}, a_{(n,2)}, \dots) \end{pmatrix}$$

$$a_{(k,l)} = \cos\left(\frac{kl2\pi}{n}\right) + i\sin\left(\frac{kl2\pi}{n}\right)), k \in \{0, 1 \dots, n-1\}, l \in \{0, 1, \dots, n-1\}$$

Často se také používá pro výpočet DFT rovnice $X_k = \sum_{n=0}^{N-1} x_n e^{2\pi\imath/Nkn}$, kde sinus a cosinus je obsažen v jednotkové kružnici vytvořené z $e^{i\varphi}$, kde na reálné ose je hodnota cosinu a na imaginární je hodnota sinu. Viz obrázek 3.2. V této rovnici x_n znázorňuje naměřená data na n-tém indexu. [56]



Obrázek 3.2 Eulerova kružnice, grafika použita z wipedia.org [57]

DFT matice je následně vynásobena vektorem vytvořeným z naměřených audio dat. Toto násobení je vlastně skalárním součinem vektorů řádků matice a skutečných dat. Tento skalární součin řádku DFT matice s naměřenými daty určuje podobnost těchto vektorů, která vyplývá ze vzorce pro skalární součin s cosinem úhlů vektorů $a \cdot \mathbf{b} = \|\mathbf{a}\| \, \|\mathbf{b}\| \cos\theta$.

PCM_q — skutečná naměřená PCM 2.6.3 data na indexu q

$PCM'_q = PCM_q + 0i$

$$\sqrt{\tfrac{1}{N}} \begin{pmatrix} A_0 \\ A_1 \\ A_2 \\ \vdots \\ A_{n-1} \end{pmatrix} \begin{pmatrix} PCM'_0 \\ PCM'_1 \\ PCM'_2 \\ \vdots \\ PCM'_{n-1} \end{pmatrix} = \begin{pmatrix} F_0 \\ F_1 \\ F_2 \\ \vdots \\ F_{n-1} \end{pmatrix}$$

Vypočítáním absolutní hodnoty komplexního čísla $|F_k| = \sqrt{[\Re(F_k)]^2 + [\Im(F_k)]^2}$ získáme výslednou podobnost k sinově křivce bez ohledu na její počáteční fázi. Při práci s reálnými daty se vyhneme složitému problému upravování dat tak, aby začínala s fází 0. Nejvyšší hodnota $|F_k|$ obsahuje nejvýraznější frekvenci, pak kf_v/n je hledanou frekvencí kde v f_v je vzorkovací frekvence a n je délka naměřených dat, které jsou předány DFT.

3.2.2 Rychlá Fourierova transformace

Rychlá Fourierova transformace (FFT) se využívá k výpočtu DFT, kde její asymptotická výpočetní časová složitost je $O(N*\ln(N))$, zatímco výpočet běžným způsobem má asymptotickou výpočetní časovou složitost $O(N^2)$. Jedná se o je zásadní rozdíl, když je počítáno s velkými daty v reálném čase.[58]

S daty o velikosti $2^{15} = 32\,768$ je výsledný počet operací s DFT řádově $(2^{15})^2 = 2^{30} = 1\,073\,741\,824$, operací. Pokud by DFT běželo na jednom vlákně (DFT by šlo paralelizovat) s daty o velikosti 2^{15} na procesoru s frekvencí 1 GHz, výpočet by trval přibližně 1 sekundu. Zatímco s použitím FFT bylo to bylo řádově $2^{15} \ln(2^{15}) \doteq 340\,696$ operací a na stejném procesoru by výpočet trval přibližně 340 µs = 0,000 34 s.[58]

Pro výpočet FFT byla použita metoda nazývaná Cooley-Tukey podle autorů James Cooley a John Tukey, kteří tuto metodu objevili a publikovali (FFT již dříve objevil Carl Friedrich Gauss v roce 1805, ale byla zapomenuta [59]). Tato metoda oproti běžné DFT vyžaduje na vstupu data o velikosti 2^x, kde $x \in 1, 2, 3 \ldots$ pro možné rekurzivní půlení problému až k datům o velikosti 1. Metoda využívá podobnosti lichých a sudých indexů na Eulerově kružnici.[58], [56]

3.2.3 Filtrování nezajímavých frekvencí

Pro kvalitnější zpracování zvuku je potřeba odstranit šum a frekvence, které pro daný úkol nejsou zajímavé a užitečné.

3.2.3.1 Okénková metoda

Pro filtrování nezajímavých frekvencí je možno využít okénkové metody. Okénková metoda využívá funkci, která modifikuje data v frekvenční doméně (až po aplikované DFT). Okénková metoda by měla mít následující vlastnosti:

- Potlačí velké množství frekvencí, které nejsou zajímavé pro daný problém.
- Frekvence nejsou příliš zkreslené v hledaném pásmu.
- Funkce efektivně přechází do hledaného pásma [56]

Pro okénkovou funkci mi, při testování s houslemi, nejlépe fungovala funkce definována následujícím předpisem:

$f(x) = 0$ pro $x \in (-\infty, 150)$

$f(x) = 1,4$ pro $x \in \langle 150, 240)$

$f(x) = 1$ pro $x \in \langle 240, \infty)$

Funkce bohužel nepřechází plynule do hledaného pásma, ale zbytek požadavků splňuje. Při testování s houslemi fungovala obstojně. (Je třeba brát na vědomí, že různé nástroje mají různé zvukové vlastnosti například různě oscilují.)

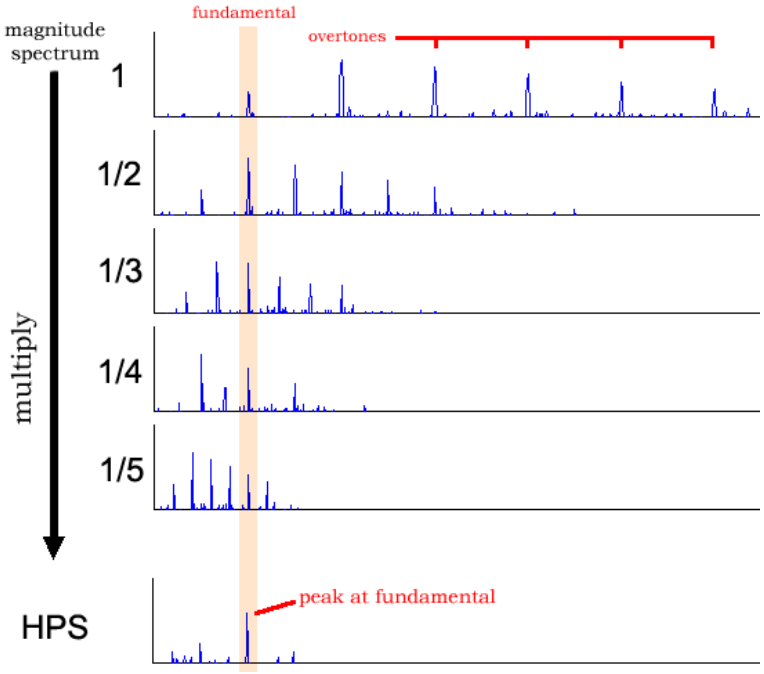
3.2.3.2 Harmony Product Spectrum

Dalším způsobem je využití Harmony Product Spectrum (HPS), které využívá oscilačních vlastnostmi hudebních nástrojů. Tato metoda lze použít pouze u nástrojů, které výrazně oscilují (například housle, flétna atd...). U neoscilujících nástrojů (či jiných zvukových zdrojů) nelze tuto metodu použít.[60]

Tato metoda opakovaně násobí původní frekvenční doménu s frekvenční doménou zmenšenou o oscilační koeficient viz Obrázek 3.3. Počet opakování je zvolen programátorem (pro housle bylo zvoleno 4 z testování). Nejběžnějším harmonickým koeficientem je jedna oktáva neboli dvojnásobek původní frekvence (u houslí najdeme oscilace v oktávách). Tyto konstantně oscilující frekvence „přežijí“ a neoscilující frekvence se vzájemně vyruší. Tuto metodu lze také využít k zpřesnění frekvence, ale v tomto projektu nebyla dostatečně přesná pro účely aplikace a použila se jiná metoda viz. Upřesnění frekvence 3.2.4. [60]

Při použití HPS je možno použít i Okénkovou metodu, ale často není třeba, jelikož šum který neosciluje v daném oscilačním koeficientu je touto metodou potlačen. Pokud bychom chtěli aplikovat HPS i okénkovou metodu, nesmíme okénkovou metodu aplikovat před použitím HPS, jelikož bychom si potlačily oscilované frekvence. Okénková metoda je v projektu použita s HPS pouze pro filtrování frekvencí, které nástroj nemůže vytvořit, žádná další logika nebyla potřeba při použití HPS.

Při vypnutí HPS je aplikována původní Okénková metoda — bez HPS a současně bez okénkové metody nelze aplikaci používat, jelikož zpracovává nesmyslné frekvence.



Obrázek 3.3 vizualizace HSP algoritmu obrázek použit z CHARM[61]

3.2.4 Upřesnění frekvence

Samotná DFT pro určení přesné frekvence v reálném čase není dostatečná, pokud není použit kvalitní mikrofon s vysokou vzorkovací frekvencí. Tyto mikrofony běžně na mobilních zařízeních nejsou.

Přesnost DFT lze zvyšovat množstvím vstupních dat. Množství dat je ale opět určeno vzorkovací frekvencí. Teoreticky lze počkat na více dat naměřených senzorem zvuku. Ale při velkém shromažďování dat k analýze už aplikace nebude reagovat plynule, jelikož se bude dlouho čekat na daná data. Data také nebudou odpovídat aktuálnímu stavu, ale budou reflektovat delší časový úsek, ve kterém daná data byla shromažďována.

Pro Výslednou frekvenci byla nejdříve použita FFT pro odhad, kde by výsledná frekvence mohla být. Následně byla použita neuniformní diskrétní Fourierova Transformace (NDFT), která umožňuje použít indexy, které neodpovídají celým číslům.[62]. NDFT bude aplikována pouze na menší interval $\langle c, d\rangle$, který byl dříve odhadnut pomocí FFT. Tímto výpočetně náročná NDFT bude aplikována pouze na velmi malý úsek dat. Díky tomu nebude ohrožena rychlost výpočtu v reálném čase a výsledná frekvence bude velmi přesná. Pro nové indexování je třeba využít menší rozdíly v indexech — α. Nová matice už nebude mít některé vlastnosti DFT. Transformační matice vypadá následovně:

$$\begin{pmatrix} A_c = (a_{(c,0)}, a_{(c,1)}, a_{(c,2)}, \dots) \\ A_{c+\alpha} = (a_{(c+\alpha,0)}, a_{(c+\alpha,1)}, a_{(c+\alpha,2)}, \dots) \\ A_{c+2\alpha} = (a_{(c+2\alpha,0)}, a_{(c+2\alpha,1)}, a_{(c+2\alpha,2)}, \dots) \\ \vdots \\ A_d = (a_{(d,0)}, a_{(d,1)}, a_{(d,2)}, \dots) \end{pmatrix}$$

$$a_{(k,l)} = \cos\left(\frac{kl2\pi}{n}\right) + i\sin\left(\frac{kl2\pi}{n}\right)), k \in \{c, c+\alpha, c+2\alpha, \dots, d\}, l \in \{0, 1, 2, \dots, n-1\}$$

3.2.5 Mapování frekvence na tón

Z výsledné frekvence je následně možno dopočítat, ke kterému tónu je frekvenčně nejblíže vypočítáním poměru zjištěné frekvence k frekvenci definované k jednotlivým notám. Na základě tohoto poměru je možno definovat vzdálenost od referenčního tónu. Nelze zde počítat vzdálenost určenou odečítáním, jelikož vnímání not je logaritmické.

V případě hudebního nástroje, který může hrát libovolnou frekvenci, která je v rozsahu nástroje (nemá před-připravené jednotlivé tóny, které může hrát jako například piáno), je potřeba kontrolovat, zda je daný tón v rámci meze, kde bude vyhodnocen jako správný a ladící. Dále je také potřeba informovat, že je daný tón správný, ale není dostatečně přesný.

Kapitola 4
Návrh aplikace

4.1 Cíloví uživatelé

Hlavními uživateli budou studenti ZUŠ, kteří hrají na hudební nástroj ve věku od 10ti do 20ti let.

Dalšími možnými uživateli můžou být lidé kteréhokoli věku, kteří budou chtít začít hrát na hudební nástroj. Aplikace ale nebude primárně cílit na požadavky těchto uživatelů v současné době. Tito uživatelé by ocenili i teoretický základ výuky, který nebude součástí této aplikace. Tito uživatelé mohou využít aplikací s kvalitně zpracovanými výukovými videi (například Trala 1.2.1)

4.2 Rozdělení obrazovek

Aplikace bude obsahovat hlavní obrazovku, kde budou základní statistiky, a bude fungovat jako rozcestník do jednotlivých částí aplikace.

Jednotlivé části budou vždy na samostatné obrazovce, aby aplikace byla přehledná a zároveň to zjednoduší vývoj — programovací logika se rozloží do více obrazovek, se kterými se bude dále lépe pracovat.

Pokud daná část aplikace bude potřebovat nějaké rozšiřující nastavení, toto nastavení bude vždy u části, které nastavuje. Při růstu aplikace by společné menu bylo rychle nepoužitelné, protože by jeho složitost byla vysoká. Uživatel by se pravděpodobně bál v aplikaci něco měnit.

4.3 Funkční požadavky

Funkční požadavky shrnou funkční části aplikace, které budou v aplikaci implementované a budou obsahovat následující seznam informací:

- Relevance — informace o důležitosti požadavku na funkčnost aplikace a jak velká priorita při následné implementaci bude mít.
  - Nízká — Požadavek není příliš důležitý v aktuálním stavu vývoje, není ani příliš důležitý pro správný chod aplikace.
  - Střední — Požadavek není příliš důležitý v aktuálním stavu vývoje, ale může výrazně zlepšit kvalitu aplikace
  - Vysoká — Požadavek je důležitý pro aktuální stav vývoje i pro samotnou kvalitu aplikace.

  - Zásadní — Požadavek je nezbytný pro samotné cíle a funkcionalitu aplikace a vývoj těchto částí by měl být prioritní před ostatními.

- Obtížnost - informace o obtížnosti při implementaci daného požadavku
  - Snadná — odpovídá odladěné implementaci v jednotkách dnů
  - Střední — odpovídá odladěné implementaci v jednotkách týdnů
  - Vysoká — odpovídá odladěné implementaci v jednotkách měsíců
- Popis - popis bude obsahovat stručný popis pro samostatnou implementaci problematiky v Androidu taky aby bylo možné vytvářet více na sobě závislých částí najednou. Také se k návrhu řešení lehčích úkolů může v budoucím vývoji využít umělá inteligence, která dokáže zpracovat menší nezávislé úkoly, při zadání kvalitního dotazu.
- Seznam souvisejících požadavků, které jsou aktuálním požadavku závislé. (Pokud na dané část nebude záviset žádný požadavek, bude vynechán)
- Module - Modul do kterého daná část bude patřit 2.5.3.

4.3.1 Ladička

Jednou z hlavních částí aplikace bude ladička, podle které si uživatel dokáže naladit svůj nástroj tak, aby pracoval správně ve zbytku aplikace, a zároveň pro samotné naladění uživatelova nástroje.

- Relevance: Zásadní
- Složitost: Střední
- Popis: Tato část je zaměřena na vykreslení Ladičky pomocí *Compose* za použití *Canvas* komponenty 2.4.2.1. Na obrazovce bude zobrazena aktuální frekvence a odpovídající nota. Pokud nota odpovídá struně na houslích, tak bude vizuálně znázorněno, o kterou strunu jde. V ladičce se následně nebudou ukazovat noty odpovídající frekvenci ale nota zvolená.
- Module: violin

4.3.2 Získání frekvence ze zvukových dat

Pro zpětnou vazbu uživateli je třeba analyzovat zvuková data a extrahovat z nich frekvenci. To je důležité ladičku a jednotlivé části aplikace, kde bude podporováno zadání noty pomocí hudebního nástroje. Výsledná frekvence musí být velmi přesná. Alespoň v rozmězí 1 (Hz)

- Relevance: Zásadní
- Složitost: Složitá
- Popis: Vstupem budou audio data ve formátu PCM 2.6.3, které budou přicházet pomocí *Flow<Array<Short>* (16bit). Výstupem bude *Flow*, ve které bude propagována nejrelevantnější frekvence nebo pokud nebude žádná frekvence dost relevantní, tak vrátí informaci oznamující že je ticho. Tato transformace bude pracovat na vláknech definovaných v *Dispatchers.Default* 2.3.2
- Závislé požadavky na tomto požadavku: Ladička 4.3.1, Rozpoznání not 4.3.8
- Module: common

4.3.3 Nahrávání zvukových dat

Aplikace bude schopna získat zvuková z mikrofonu zařízení.

- Relevance: Zásadní
- Složitost: Snadná
- Popis: Úkolem bude vytvořit třídu, která bude na množině vláken definovaných v *Dispatchers.IO* 2.3.2 provádět čtení Audio dat z mikrofonu ve formátu *PCM-16bit* pomocí *AudioRecord* třídy. Při nahrávání audia musí třída kontrolovat, zda má oprávnění číst z mikrofonu. Výstupem bude *Flow<Short>* audio dat.
- Závislé požadavky na tomto požadavku: Detekce zvuku 4.3.6, Získání frekvence ze zvukových dat 4.3.2
- Module: common

4.3.4 Metronom

Aplikace by měla obsahovat metronom tj. běžně používané časovací zařízení pro asistenci hráče na hudební nástroj držení se v rytmu.

- Relevance: Zásadní
- Složitost: Střední
- Popis: Aplikace by měla obsahovat obrazovku s Metronomem, kde půjde nastavit rychlost metronomu i počet dob v taktu. Aplikace bude znázorňovat doby jak vizuálně tak zvukově. První doba bude odlišena jak jiným zvukem, tak i graficky.
- Module: common

4.3.5 Opakování rytmu

Aplikace bude obsahovat část, kde uživatel bude opakovat zadaný rytmus zvukem.

- Relevance: Vysoká
- Složitost: Vysoká
- Popis: Na jedné obrazovce zařízení se bude přehrávat rytmus a bude vyžadováno, aby uživatel zadal rytmus, buď pomocí hry na hudební nástroj, tleskáním (hudební nástroj zde není zásadní) nebo klepáním na obrazovku.
- Module: common

4.3.6 Detekce zvuku

Aplikace by měla detekovat, kdy je přítomen hluk a kdy není pro možnost tvorby funkcionalit založených na rytmech.

- Relevance: Vysoká
- Složitost: Střední

- Popis: Tato část aplikace bude přijímat zvuková data ve formátu PCM 2.6.3 ve *Flow* a výstupem bude *Flow*, která určí zda je hluk nebo není
- Module: common
- Závislé požadavky na tomto požadavku: Hraní rytmů podle not 4.3.7, Opakování rytmu 4.3.5

4.3.7 Hraní rytmů podle not

Uživatel bude dostávat zadaný rytmus v notovém zápisu a bude ho muset opakovat.

- Relevance: Vysoká
- Složitost: Střední
- Popis: Podobný úkol jako opakování rytmů4.3.5. Zde pouze není definován rytmus pomocí zvukových výstupů aplikace, ale vizuálně podle notového zápisu.
- Module: common

4.3.8 Rozpoznání not

Aplikace bude zobrazovat náhodné noty, uživatel následně bude muset zahrát odpovídající tón.

- Relevance: Zásadní
- Složitost: Střední
- Popis: Obrazovka na které bude propojeno vykreslování not, zadávání pomocí hmatníku a samotné rozpoznávání hrané frekvence. Obrazovka zobrazí náhodnou notu v notovém zápisu a následně bude očekávat uživatelovo zadání jak zvukově, tak pomocí vykresleného hmatníku.
- Module: violin Zadávání Pomocí Hmatníku 4.3.11

4.3.9 Vykreslení not

Vykreslení not umožní komunikaci s hudebníkem ve mnoha různých dalších požadavcích.

- Relevance: Zásadní
- Složitost: Složitá
- Popis: Cílem je vytvořit systém vykreslující notové zápisy, který bude správně pracovat s předznamenáním, posuvkami a dokáže se adaptovat, pokud bude zadán delší počet not než se dokáže zobrazit na obrazovce. Postupovat se bude z 3.1
- Závislé požadavky na tomto požadavku: Rozpoznání not 4.3.8, Hraní rytmů podle not 4.3.7
- Module: common

4.3.10 Možnost nastavení barev not

Rozšíří Vykreslení Not 4.3.9 o možnost nastavení barev jednotlivým notám, které si bere za cíl snazší práci uživatelů s notami.

- Relevance: Střední
- Složitost: Snadná
- Popis: K notám bude předáno mapování na barvu. Toto obarvení bude moci uživatel upravovat a bude implementováno mimo samotnou implementaci Not.
- Závislé požadavky na tomto požadavku: Vykreslení Not 4.3.9 (pro správný chod vykreslení not není klíčový)
- Module: common

4.3.11 Zadávání pomocí hmatníku

Uživatel bude schopen zadat vstup do aplikačních částí pomocí vykresleného hmatníku, aby mohl v aplikaci používat i v prostorech, kde není možné použít nástroj samotný. Například cestou domů v autobuse.

- Relevance: Zásadní
- Složitost: Střední
- Popis: Toto zadání hmatníkem bude součástí jednotlivých úkolů, kde se zadává zvuk pomocí nástroje. Přidá se možnost zadat odpovídající tón pomocí vykresleného hmatníku, kde jednotlivé prsty budou očíslovány pro větší přehlednost tohoto hmatníku. Prstoklady na hmatníku také musí odpovídat správné tónině, ve které je úkol zadán. Tato informace o tónině bude předána hmatníku parametrem.
- Module: violin
- Závislé požadavky na tomto požadavku: Rozpoznání not 4.3.8

4.3.12 Bodový systém

Primární motivační prostředek, který by měl motivovat uživatele se vrátit zpět k aplikaci.

- Relevance: Zásadní
- Složitost: Střední
- Popis: Bude vytvořen bodový systém, který bude odměňovat uživatele. Tento bodový systém musí motivovat uživatele pro hraní na nástroj vždy, kdy to je možné. Tedy bodový zisk bude navýšen při zadávání jednotlivých úkolů pomocí nástroje. Bodový systém bude pro širší možnosti práci s daty uložen v *SQLite* úložišti 2.4.5.
- Module: common
- Závislé požadavky na tomto požadavku: Rozpoznání not 4.3.8, Opakování rytmu 4.3.5, Hraní rytmu podle not 4.3.7, Využití bodů pro grafické odměny na vlastním avataru (postavičce)4.3.13

4.3.13 Využití bodů pro grafické odměny

Tato část by měla fungovat jako hlavní motivační prostředek hráčů, kde své nasbírané body. Budou vyměňovat za grafické úpravy svého hrdiny s nástrojem.

- Relevance: Nízká
- Složitost: Vysoká
- Popis: Pomocí získaného skóre bude možné si vytvářet a modifikovat vlastní postavičku s nástrojem. Počet mincí, které uživatel bude vlastnit nebude přímo počítáno ze skóre jelikož při nákupu by se mu skóre snížilo a to by rozbilo statistiky.
- Module: common

4.3.14 Ukládání a načítání not do MusicXML

MusicXML je formát pro ukládání notových zápisů. Možnost uložit a načít tyto data dokáže propojit již existující programy (například pro tvorbu notových zápisů) s touto aplikací.

- Relevance: Střední
- Složitost: Vysoká
- Popis: Aplikace převede interní reprezentaci notových zápisů do formátu MusicXML a uloží jej do souboru. V aplikaci bude existovat i opačný převod, kdy souborve formátu MusicXML převede na interní reprezentaci not.
- Module: common

4.4 Nefunkční Požadavky

Nefunkční požadavky míří hlavně na strukturu projektu technologie a další prvky, které nejsou spojené s funkční stránkou aplikace. Bude obsahovat pouze informace o relevanci a složitosti stejné z 4.3, pokud dané parametry nebudou mít smysluplnou hodnotu, budou nahrazeny „—“.

4.5 Jediný zdroj pravdy v aplikaci

Zajistí snadnou rozšiřitelnost aplikace do šířky (nové nezávislé funkcionality bude snadné přidat).

- Relevance: Zásadní
- Složitost: —

4.5.1 Statická analýza

Upozorní na možné problémy s kvalitou kódu.

- Relevance: Zásadní
- Složitost: —

4.5.2 Automatizace statické analýzy a testů

Provede testy a statickou analýzu externě pomocí CI/CD.

- Relevance: Vysoká
- Složitost: Snadná-Střední (závisí na potřebách CI/CD)

4.5.3 Zajistit 95% kompatibilitu s existujícími mobilními zařízeními

Tato kompatibilita zajistí, že aplikace bude dostupná téměř všem uživatelům aplikace.

- Relevance: Zásadní
- Složitost: Snadná

4.5.4 Optimalizace *Compose*, při práci se zvukem

Aplikace bude při změně zvuku aktualizovat *Compose* komponenty na obrazovce, které souvisí se zobrazením informace získané ze zvukového stavu. Ostatní komponenty se nebudou překreslovat. Toto zajistí plynulost aplikace i při časté změně stavů pracujících se zvukovými daty.

- Relevance: Zásadní
- Složitost: Snadná

4.5.5 Aplikace bude používat verzovací katalog

Pro snazší práci s více moduly bude využit verzovací katalog 2.6.2.

- Relevance: Zásadní
- Složitost: Snadná

4.5.6 Memorizace nastavení aplikace

Informace nastavené uživatelem budou opět přístupné při dalším otevření aplikace.

- Relevance: Zásadní
- Složitost: Snadná

4.5.7 Dokumentace

V android vývoji jsou velmi vítány dokumentační komentáře přímo v kódu. Uživatelé nejsou pak nuceni tak často otevírat externí programy pro zobrazení dokumentace. Také je často aktuálnější, jelikož se změnou se aktualizuje hned a nemusí se následně přepisovat do externích programů nebo generovat z dokumentačních komentářů.

- Relevance: Vysoká
- Složitost: —

Kapitola 5

Popis vývoje a testování funkčnosti

Tato kapitola se zaměří na popis samotného procesu vývoje a postupu při testování aplikace. Také ke konci popíše další vývoj aplikace.

5.1 Výběr cílů implementace

Z analýzy požadavků bylo zjištěno, že jednotlivé požadavky jsou náročné, proto při implementaci byl kladen důraz hlavně na nefunkční požadavky, které zjednoduší další rozvoj aplikace a zásadní funkční požadavky.

Z nefunkčních požadavků byly zvoleny požadavky na dokumentaci, statickou analýzou, DI a jediný zdroj pravdy, kterými hude zajištěno, že aplikace bude snadno dále rozšiřitelná a snadno udržitelná v budoucím vývoji. Jednotlivé části také budou použitelné i v jiných projektech. Nebude se tedy muset stále dokola implementovat a testovat jedna a ta samá funkcionalita.

5.2 Statická analýza kódu

Pro jednotný a čistý kód byla využita statická analýza kódu pomocí nástroje *Detekt*. Tato analýza sjednotí všechen kód. Zajistí, že programátor dokumentuje a dodržuje pravidla jazyka, který používá. Nevytváří dlouhé metody / funkce. Dělí kód do tříd tak, aby jedna třída nedělala veškerou logiku aplikace, udržuje jmenné konvence jazyka (například názvy funkcí začínají malým písmenem třídy velkým), jednotlivé funkce nemají nesmyslně vysokou cyklomatickou složitost a další…

V nastavení Detektu jsem navíc nastavil(*Detekt* obsahuje v základu, pokud není nastaveno jinak, nastavení, které míří na většinu aspektů jako jsou dlouhé třídy magické konstanty a další…), pomocí nastavení v souboru *detekt.yaml*, že každá veřejná (*public*) třída musí mít dokumentační komentář a tím byla zajištěna dokumentace přímo v kódu. Pro statickou analýzu *Composable* funkcí musela být nastavena speciální pravidla, jelikož s *Composable* funkcemi se pracuje odlišným způsobem než s běžnými funkcemi. Stejná pravidla byla nastavena jak pro modul *violin*, tak pro modul *common*.

Detekt následně vytváří hlášení s chybami (pokud jsou přítomny), které se v kódu vyskytují. Také hodnotní kód informacemi jako je cyklomatická složitost, počet řádků kódu a další…. Veškeré tyto informace generuje v jednom ze zvolených formátů. V tomto projektu jsem zvolil, aby se generovaly výstupy této analýzy ve formátu HTML. Výsledný výpis Detektu viz A.1

5.3 Verzovací katalog

Pro snazší práci s více moduly a jejich závislostmi byl vytvořen verzovací katalog TOML 2.6.2, ve kterém byly definovány jednotlivé závislosti strukturovaně. Tento soubor byl uložen v *./gradle/libs.versions.toml*. Pokud je tento soubor uložen přímo na této adrese, není třeba ho v samotném sestavovacím skriptu explicitně definovat.

Následný TOML soubor byl rozdělen na jednotlivé části, kde každá část je označena jedním z následujících identifikátorů:

`[versions]` označuje část, kde jsou uloženy jednotlivé verze, které se aplikují opakovaně (není třeba zde mít zapsané verze, které se aplikují pouze jednou, tyto verze je možno přímo zapsat k jednotlivým částem, kam patří).

`[libraries]` označuje část, kde byly uloženy všechny knihovny v projektu. Zde se těmto knihovnám nastavila verze buď staticky, nebo referencí na verzi z části označené `[versions]`.

`[boundles]` označuje část, kde jsou uloženy sady knihoven. Tyto knihovny jsou spojeny do jednoho balíčku a je možno je v sestavovacím skriptu aplikovat pomocí jediného příkazu *implementation*.

`[plugins]` označuje část rozšiřujících pluginů pro sestavovací skript

Ukázka verzovacího katalogu viz. A.5 a aplikace v sestavovacím skriptu viz. A.4

5.4 Memorizace obrazovek

Memorizace obrazovek byla provedena pomocí ukládání jednotlivých stavů do *SharedPreferences* 2.4.5. Občas však bylo potřeba uložit složitější objekty než pouze základní typy, které pomocí běžného použití *SharedPreferences* nelze uložit.

Pro uložení složitějších struktur byl vytvořen DataStore s vlastním serializačním objektem, který ukládaná data převedl do JSON formátu, který bylo možné následně uložit jako klasický soubor. Při čtení se opět převedl JSON objekt ve *String* na původní objekt v programu. Pro čtení nastavení, které se vždy musí přečíst celé a není nijak strukturované, mi přišlo nesmyslné vytvářet pro tyto potřeby záznamy v *SQLite* databázi.

Pro serializaci dat do a z JSON *String* byla použita knihovna pro Kotlin pro serializaci do JSON objektů. Samotný převod na Json nebyl třeba explicitně řešit pouze ho bylo nutno zapojit do serializačního objektu, který byl předán DataStore knihovně. Dále bylo potřeba označit příslušný objekt (popřípadě všechny podobjekty) anotací *@Serializable*.

5.5 Zpracování zvuku

Klíčovým bodem funkční implementace aplikace bylo vyřešení náročného úkolu spočívající v analýze zvuku v reálném čase, protože tato funkce tvoří základ funkčnosti aplikace, od které je závislý další vývoj jednotlivých částí. Abych tento úkolu vyřešil, začal jsem výzkumem využití DFT 3.2.1, pro analýzu zvuku v reálném čase na zařízeních se systémem Android. Zjistil jsem však, že samotný algoritmus DFT není dostatečně výpočetně rychlý, aby dokázal analyzovat naměřená data v reálném čase. Následně jsem se rozhodl prozkoumat FFT 3.2.2 jako alternativu.

Ačkoli FFT nabídla výrazné zlepšení z hlediska výpočetní rychlosti, měla výrazný problém v určení přesné frekvence. Jako logický postup mi přišlo zvýšit vzorkovací frekvenci zařízení. Následně jsem zjistil, že zařízení s Androidem 10 a výše musí podporovat takzvané *High-resolution* audio, které umožňuje nastavit vzorkovací frekvenci na 192 000. Bohužel ani s *High-resolution* audiem, jsem nebyl schopen získat frekvenci dostatečně přesnou. Odchylka byla řádově 10 Hz. Pro zvýšení přesnosti frekvenční analýzy jsem pak experimentoval s algoritmem *Harmonic Product*

Spectrum (HPS). HPS sice pomohl vyčistit frekvenční oblast, ale výsledná frekvence stále nesplňovala požadovanou úroveň přesnosti. Tato metoda s kombinací *High-resolution* audio přinesla přesnost v rozmezí 2 Hz.

Nakonec jsem vyvinul vlastní přístup kombinací FFT k vymezení prostoru, kde by výsledná frekvence mohla vyskytovat a následnému upřesňujícímu algoritmu, který se ukázal, že je založen na NDFT. Toto vlastní řešení poskytlo potřebnou kvalitu výkonu a přesnosti a umožnilo aplikaci efektivně analyzovat zvukový vstup v reálném čase s přesností až na 0,05 Hz.Tato přesnost je více než dostačující na kvalitní analýzu frekvence zvuku v aplikaci.

High-resolution audio se vzorkovací frekvencí 192 000 jsem v aplikaci zatím ponechal, přestože v dokumentaci píší, že je vyžadováno až od Androidu 10.0 (API 29). Aplikace mi fungovala i na zařízení s Androidem 9.0 a na Emulátoru s Androidem 9.0. Při zjištění častého padání aplikace pomocí *Firebase* na konkretních zařízení by bylo možné tento problém dále řešit snížením vzorkovací frekvence a zmenšením délky čtených audio dat. Některé zařízení je možné aktualizovat na novější verze, tak by třeba uživatelům stačilo pouze aktualizovat systém.

Avšak pochopení HPS nebylo zbytečné, protože se v následném testování s houslemi projevilo nejlepší při filtrování nezajímavých frekvenčních hodnot. Toto zapojení HPS je možné snadno nastavit a vypnout pomocí změny v DI zodpovědné o samotnou transformaci. Pro testování generátorem frekvencí je dobré ho vypnout, protože generátor frekvencí generuje pouze původní frekvenci (bez oscilací).

Ovládání třídy SingleFrequencyReaded je vytvořeno tak, že uživatel pouze při začátku použije metodu *SingleFrequencyReaded.start()*, která spustí samotné nahrávání PCM audio dat ve třídě *PcmAudioRecorder*. Po ukončení stačí zavolat funkci *SingleFrequencyReaded.stop()*. Samotná data o frekvenci putují z *PcmAudioRecorder* do *SingleFrequencyReaded* pomocí *Flow<ShortArray>* 2.3.2.1 a následně je výsledná frekvence prezentována (Observer pattern) v *SingleFrequencyReaded.frequency* jakožto *Flow<Double>* výsledné frekvence, které umožní sledovat hodnotu frekvence v reálném čase.

5.6 Vykreslení not

Při hledání vhodné knihovny pro vykreslování notových záznamů v aplikaci jsem našel jedinou knihovnu, která by se dala použít , ale byla napsána v programovacím jazyce JavaScript, a dala se zprovoznit pomocí *WebView* pro zobrazování webových stránek. Tento přístup mi nepřišel vhodný jak z výkonnostního hlediska, tak z další možné rozšiřitelnosti a čitelnosti zdrojového kódu. Proto jsem se rozhodl vytvořit vlastní řešení pro vykreslování not pomocí *Compose* a jejího rozhraní *Canvas*. *Composable* funkci zodpovědnou pro vykreslení notových zápisů jsem pojmenoval *Sheet*. Dále jsem vytvořil jednotlivé funkce pro vykreslování jednotlivých prvků ze základních tvarů (ovály, čáry…), jako jsou noty, klíč (klíč byl na rozdíl od zbytku vytvořen grafickým editorem a vykreslen do Canvas jako vektorová grafika), křížky, béčka, které jsem poté zkombinoval do notového zápisu. Tento přístup umožnil větší flexibilitu a přizpůsobení notového zápisu.

Tato *Composable* funkce *Sheet* má rozhraní, ve kterém se jí předá list not a *Sheet* sám rozhodne, jak má vykreslit jednotlivé noty a kolik řádků má vykreslit. Uživatel nemusí složitě přemýšlet, kolik místa musí nechat, aby se tato Composable komponenta správně vykreslila. Jediné, co musí uživatel vědět, jsou samotné noty, které potřebuje vykreslit. Toto rozhraní umožňuje použití této *Composable* funkce v mnoha různých částech aplikace i v jiných aplikacích. proto jsem také tuto komponentu zařadil do společného modulu *common*.

Samotné noty jsou definovány vlastní třídou *SheetNote*, která obsahuje informaci o notě v notovém zápisu i její délku trvání, přidělenou posuvku, a další… Určitá nota je určena názvem a oktávou, kde pro uložení názvu je použit výčet hodnot (enum).

Aktuální verze zatím nedokáže vykreslit všechny délky not (dokáže od celé do šestnáctinové), taktové čáry, také většinu speciálních znaků. Tyto prvky nejsou v aktuální verzi aplikace potřeba.

5.7 Reprezentace tónů

Jednotlivé tóny jsou v aplikaci definovány třídou TweltoneTone, kde v parametru je nastaven název určeným výčtem hodnot (enum *InnerTwelveToneInterpretation*) a oktávou. Těmto tonům je při startu aplikace potřeba nastavit jmenné mapování pomocí metody *TwelveToneNames.setTwelveTonesNames()*, jelikož v různých zemích se stejné noty nazývají jinými názvy (například české H je anglické B). Tyto tóny jsou následně rozšířené (dědičností) o tóny s frekvencí. Aplikace totiž umožňuje upravit výchozí frekvenci tónu A4, podle které se následně určí všechny ostatní tóny na základě výpočtu tónů v 12-ET viz. 3.1.

5.8 Určení tónu hraného hráčem

Pro určení tónu hraného hráčem je nalezeno minimum absolutní hodnoty rozdílu mezi číslem 1 a poměrem výchozí frekvence získané z *SingleFrequencyReader* k referenční frekvenci obsažené v *TwelvetoneTone*. Z tohoto rozdílu lze také odvodit zda tón hraný hráčem ladí, nebo neladí. Pro větší flexibilitu těchto výpočtů je přesnost tohoto ladění oddělena v nastavení samotného tónu s frekvencí pomocí parametru inTuneInterval a inRangeInterval. Tyto intervaly byly specifikovány výčtem, kde jsou definovány základní hodnoty těchto přesností typem (*LOW, MEDIUM, HIGH*). Tato specifikace výčtem je přehlednější než nastavení přímými hodnotami.

Jelikož v ladičce je potřeba přesnější ladění než v samotných hrách, kde je lepší dát větší prostor na chybu (není třeba natolik přesně zahraný tón jako v ladičce na vyhodnocení že je správný). V ladičce byla nastavena přesnost na *InTuneInterval.HIGH*, která indikuje ladící tón, když je jeho absolutní hodnota rozdílu menší než 1/20 rozmezí jednotlivých tónů definovaných v 12-ET. Ve hrách je nastaveno mírnější hodnocení *InTunePrecision.MEDIUM* odpovídající maximálnímu rozdílu 3/20 mezi jednotlivými tóny.

Aby bylo ve hrách zajištěno, že hudebník hraje správné noty musel být zajištěn převod not na tóny. Pro tento převod jsem vytvořil extension funkce 2.3.1, které zajišťují tuto logiku. Pro převod mezi notou a tónem musely být také brány v potaz jednotlivé posuvky, které posouvají tón na 12-ET stupnici.

5.9 Skóre

Skóre v jednotlivých hrách bylo uloženo do SQLite databáze pomocí knihovny *Room* 2.4.5. Tato knihovna nebyla použita samostatně ale spolu s generovanou částí pomocí *ksp* generátoru kódu. Pro samotné uložení dat jsem vytvořil třídu *ScoreDao* označené anotací *@Dao*. V *ScoreDao* jsou označeny jednotlivé funkce anotacemi *@Query* ve kterých jsou definovány příslušné SQL dotazy, popřípadě *@Insert* pro vložení předmětu do databáze bez specifikace přesného SQL dotazu. Dotazy opět vrací data součástí *Flow*, které zaručí reaktivní aktualizace těchto hodnot (při změně hodnoty v DB se změna propíše do výsledku). Dále zde bylo potřeba vytvořit objekt popisující samotnou tabulku v databázi anotací *@Entity* a v něm anotací *@PrimaryKey* označen primární klíč entity. V parametru anotace *@PrimaryKey* bylo označeno, že při vkládání se *id* generuje automatiky. Jinak by bylo třeba udržovat informaci o posledním *id* někde v programu. To je ale zbytečně složité.

5.10 Obrazovky

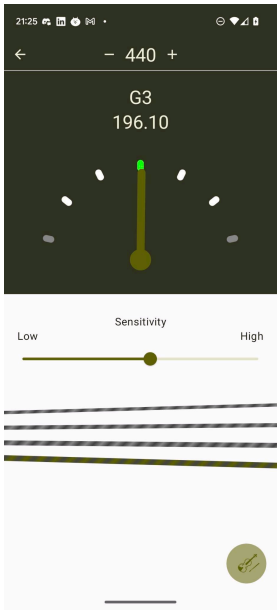
Tato část provede čtenáře jednotlivými obrazovkami aplikace implementovanými v frameworku *Compose UI* 2.4.2 za použití MVVM architektury 2.5.2.

5.10.1 Hlavní obrazovka

Hlavní obrazovka byla navržena jako rozcestník mezi jednotlivými částmi aplikace a zároveň obsahuje informace o celkovém skóre a denním skóre. Je nastavena jako úvodní destinace navigace pomocí anotace *@RootNavGraph(start = true)*, která zajistí že při vytvoření navigace bude navigováno právě na tuto destinaci. V *MainActivity* bylo pouze nastaveno *Compose UI* s hlavní navigací, která byla vygenerovaná knihovnou *Destinations* 2.4.2.2.1. Ostatní cíle navigace byli označeny pouze anotací *@Destination*

5.10.2 Ladička

Obrazovka ladičky čte data z třídy *SingleFrequencyReader* a mapuje je na konkrétní tón pomocí objektu *FrequencyToTone*. Samotný stav na obrazovku je uložen v *ViewModel* třídě, který zajistí přežití stavů i při překreslení aktivity. Tento stav je prezentován pomocí *StateFlow* na obrazovku, kde se vykresluje pomocí *Canvas* funkce budík s ručičkou, která ukazuje jak moc přesně je tón hrán 5.8. Tento úhel ručičky se počítá pomocí funkce *TwelvetoneTone.getDifferenceAngle*, která vrátí násobek vzdálenosti od původního tónu rozšířen na úhel a předán v parametru této funkci. Budík pro snazší práci při vykreslování byl vykreslen uprostřed obrazovky a následně rotován podle středu. Výsledný obraz budíku je pouze posunut do místa, kam patří. Dále je na horní části obrazovky přidána informace o tónu, který odpovídá, i s naměřenou frekvencí. Ve spodní části jsou vykresleny jednotlivé struny, které se rozsvítí, pokud je hrán tón odpovídající konkrétní struně. Červeně, pokud neladí, a primární barvou (z *MaterialTheme*), pokud ladí. Dále byla přidána funkcionalita pomocí Composable funkce *ScaffoldFloatingButton* tlačítkem z Material3 knihovny *FilledIconToggleButton*, kterým je možno nastavit, zda se mají ukazovat v ladičce všechny tóny, nebo pouze tóny strun nacházejících se na houslích.



Obrázek 5.1 Obrazovka Ladičky (v obrázku také možno vidět využití Dynamických barev — Žlutá)

5.10.3 Rozpoznání tónu

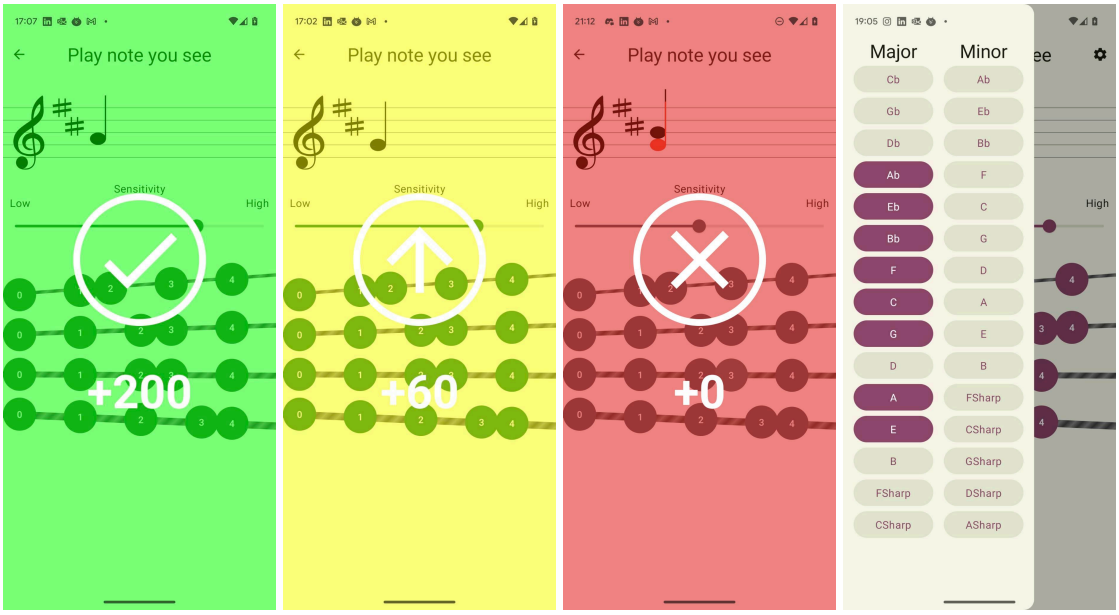
Další obrazovkou aplikace byla vytvořena část pro rozpoznání tónů, kde do notového zápisu *Sheet* je vygenerována náhodná nota. Tu uživatel musí následně zahrát na hudební nástroj za zisk 200 bodů za tón, který ladí, 60 bodů za tón který je správný, ale není dostatečně přesný a 50 bodů pokud uživatel zadal tón pomocí vykresleného hmatníku. Vyhodnocení jednotlivých vstupů viz. 5.2

Hmatník jsem vykreslil pomocí nativního vykreslení pomocí *Composable* funkce *Canvas*. Uživatel může zadat zobrazenou notu i bez nutnosti hrát na hudební nástroj. Vykreslený hmatník respektuje prstoklad aktuální tóniny, ve které je vygenerovaná nota.

Dále jsem zde přidal možnost výběru tóniny ze všech existujících tónin (bez duplikovaných). Tento výběr byl přidán do *Drawer* komponenty, která lze otevřít tlačítkem v horním menu nebo přejetím prstem po obrazovce. Pokud je vybráno více tónin, vybere se náhodná z tohoto výběru. Výběr tónin poslední obrazovka v 5.2

Po zadání noty (jakýmkoli způsobem) se ukáže zda uživatel zadal správnou notu (čistě nebo falešně s ukazatelem, kam má posunout prst aby byla správně) nebo špatnou notu. Pokud zadal notu špatně, zadaná nota se mu vykreslí do notového zápisu tak, že vidí najednou původní správnou notu i jeho špatně zahranou (špatně zahraná je odlišena chybovou barvou z *Material Design*. Zde je problém s tím, že barva pro chybu a primární barva v *Material Design* můžou být obě červené. Pak není jednoduché tyto barvy rozlišit). Následně po 3sekundách se vygeneruje nová nota a uživatel může zadávat znovu.

Jelikož k této části aplikace není vždy potřeba přístup k mikrofonu, nastavil jsem, aby obrazovka fungovala i bez oprávnění nahrávat zvuk — aplikace nezačne nahrávat audio, ale vstup vykresleným hmatníkem je možný. Také jsem přidal možnost vypnout nahrávání zvuku, i když má uživatel povoleno oprávnění nahrávat zvuk, aby mu v rušném prostředí nevyhodnocoval nesmyslné zvukové vstupy.



Obrázek 5.2 Vyhodnocení zadání tónu a nastavení stupnic

Při první notě obdržené z *SingleFrequencyReader* může být přečtený tón chybný, jelikož hraný tón je obsažen pouze v malé části načtených dat. Proto se vyhodnocení, zda uživatel hraje správný tón, musí použít až pokud je po sobě přečten dvakrát stejný tóny z *SingleFrequencyReader*. Samotná přesnost je určena až z druhého tónu.

5.10.4 Metronom

Metronom byl vytvořen časovačem, který opět pomocí *Flow* aktualizuje obrazovku. Tento časovač běží na vlastní *Coroutine* a pro zastavení se *Coroutine* zastaví funkcí *Job.cancel()*. Zároveň se stavem obrazovky jsou přehrávány zvukové záznamy odpovídající skutečnému metronomu pomocí *MediaPlayer* třídy. Této třídě je předáván soubor ve formátu MP3. V metronomu jsem přidal možnost upravovat rychlost úderů v jednotce *Beats per minute* (BPM). A také možnost nastavit počet dob v taktu. Zvuk první doby byl nastaven odlišným zvukem, pro rozpoznání první doby.

Signalizace metronomu barvou obrazovky v aktuální implementaci není moc příjemná, nejspíš tato funkce bude úplně odstraněna. Nebo se bude muset promyslet jiný způsob signalizace v aktuální implementaci, je tato signalizace velmi nepříjemná.

5.11 Testování

Pro testování naměřená frekvence jsem zpočátku používal testování s houslemi (dvěma různými). Toto testování bylo sice věrohodné, ale nebylo příliš efektivní a přesné. Byl jsem však schopen zjistit, zda naměřena frekvence je správná, jelikož jsem si předem nástroj naladil. Při testování jsem pak zkoušel mírně rozlaďovat nástroj dolů a nahoru a tyto změny jsem následně očekával na výstupu z zařízení. Toto testování však nebylo velmi přesné a nebylo příliš efektivní.

Následně jsem přešel na testování s generátorem frekvencí na mobilní zařízení [63]. Jedním mobilním zařízením jsem generoval zvuk o zvolené frekvenci a druhým jsem opět tuto frekvenci měřil. Při tomto testování s generátorem frekvencí musela být vypnuta HPS transformace, jelikož generátor nevytvářel oscilující frekvence viz. A.2 oproti houslím A.3. Pro samotnou spektrometrii byla využita webová stránka [64]. Tato frekvence z generátoru sice neodpovídala reálnému využití, ale frekvence byla velmi přesná a bylo možné vytvářet libovolné frekvence velmi efektivně a přesně.

Následně některé části, ve kterých neočekávám velké změny a ve kterých jednotkové testy měly smysluplnou výstupní hodnotu, jsem pokryl jednotkovými testy pomocí knihovny *JUnit Jupiter* pro testování a *Mockk* pro reflexivní náhradu částí kódů za testovací implementace.

Pro testování aplikace byli využity primárně zařízení (novější) Google Pixel 7 s API 33 a (starší) Xiaomi Redmi 9 s API 30.

5.12 Continuous integration

V poslední řadě jsem nastavil automatické testování pomocí Unit testů a automatickou statickou analýzu pomocí Continuous integration (CI). Toto CI bylo nastaveno v *GitLab* repozitáři, kde při každé další změně v projektu (commitu) (pokud někdo pomocí toho commitu nesmaže nastavovací soubor) se provedou Unit testy, statická analýza kódu, a sestaví se *debug* verze aplikace automaticky.

Sestavení *release* verze na serveru bohužel trvalo příliš dlouho a po 20minutách server toto sestavení násilně ukončil. Při sestavení *release* verze byly nastaveny minimalizace a optimalizace při kompilaci.

Jelikož projekt obsahuje nové technologie nebyl nalezen odpovídající docker image (docker image obsahuje základní nastavení virtuálního stroje, na kterém se provádí samotné CI). Proto musela být manuálně doinstalována java 19, aby bylo možné spustit jednotlivé části.

5.13 Další vývoj aplikace

Pro práci s požadavky od uživatelů bych volil agilní přístup, který nejlépe reaguje na změny požadavků uživatelů, ale vyžaduje častou zpětnou vazbu od zákazníků, která aktuálně roste s postupem v projektu.

5.13.1 Probrání nápadů mladých hudebníků

Pro kvalitnější motivaci hudebníků by bylo potřeba projít tuto problematiku přímo s dětmi, pro které je tato aplikace určená.

5.13.2 Uživatelské testování aplikace

Aplikace by měla projít několika různými uživatelskými testy, které by měly cílit na spokojenost jak ze strany učitelů hudby, tak i ze strany samotných žáků. A identifikovat problémy s UI.

Pro podporu tohoto testování by měla být také zapojena služba Firebase, pomocí které je možné shromažďovat data jednotlivých uživatelů, na základě kterých lze sestavit statistiky, které mohou indikovat problémy a funkční nedostatky spojené s používáním aplikace (ne pouze chyby aplikace).

5.13.3 Onboarding

Vytvořit srozumitelný onboarding (onboarding označuje úvod do aplikace, který by měl vysvětlit jak pracovat s aplikací), který zajistí příchod a udržení nových uživatelů. V tomto úvodu by měly být jednoduše nejlépe za pomocí obrázků či videí vysvětleny veškeré důležité funkcionality aplikace a zároveň jak tyto funkcionality použít v praxi.

5.13.4 Backend

Vytvořit backend část aplikace, která zajistí vytvoření uživatelských účtů a následné možnosti práce s více uživateli. Také přinese možnost vytvářet žebříčky pro další motivaci uživatelů.

5.13.5 Průzkum dalších technologií pro zpracování audio dat

Prozkoumání dalších transformačních metod pro práci s audio daty. Kterou je například *Discrete wavelet transform* nebo rychlou verzi NDFT. *Discrete wavelet transform* by mohla být užitečná při rozpoznání při identifikaci hraných rytmů. Nebo při kontrole rychlých změn v nahraném zvuku.

5.13.6 První vydání aplikace

Vydat aplikaci na Google Play Store. Dále nastavení CI/CD pro zjednodušení práce s vydáváním aplikace. Při tomto propojení je také možné aplikovat zjednodušené přihlášení do aplikace pomocí Google Play Store účtu. Toto vydání ale bude muset být až po provedení uživatelského testování a implementování všech důležitých částí aplikace.

Kapitola 6
Závěr

V této práci bylo úkolem navrhnout a implementovat prototyp mobilní aplikace pro zřízení s operačními systémy Android, který si bere za cíl ukázat možný směr budoucím aplikacím, které by mohly pozitivně ovlivnit výuku hry na hudební nástroj.

V 1. kapitole byl proveden průzkumu existujících řešení a konzultace s pedagogy v oboru. Z tohoto průzkumu vyplynulo, že uživatele by motivovala aplikace, která by s nimi aktivně spolupracovala v reálném čase.

Následně v 2. kapitole byly vybrány a popsány technologie použité pro samotnou realizaci mobilní aplikace. Tyto technologie byly cíleny na efektivitu práce s daty na zařízeních Android a reaktivní způsob programování pro plynulost aplikace.

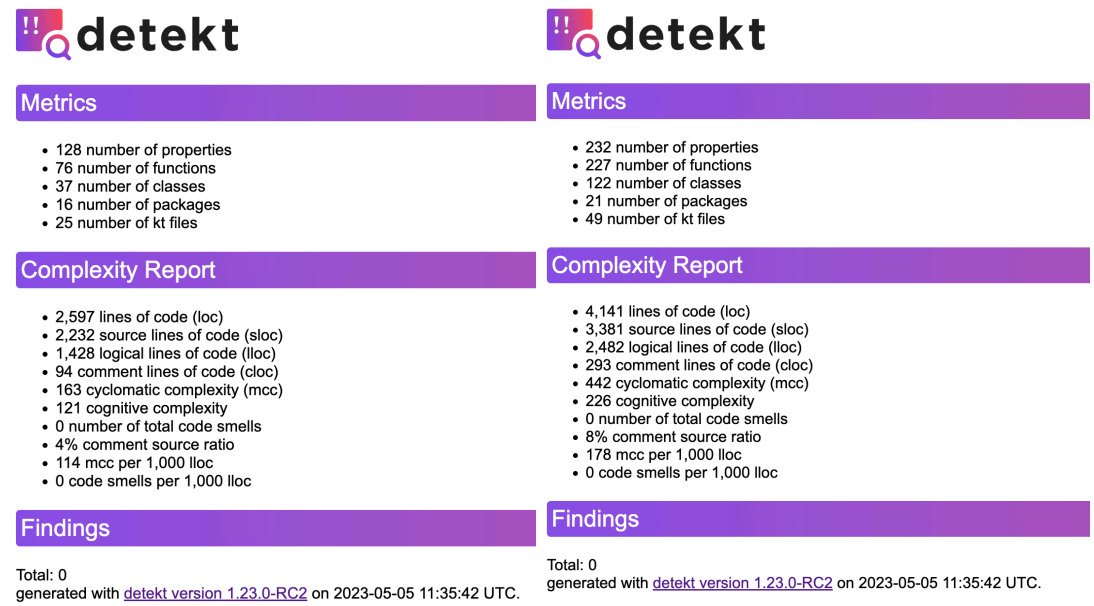
Ve 3. kapitole byl prozkoumána problematika spojená s reprezentací not a vlastnosti tónů. Dále zde byl prozkoumán potup pro získání frekvence v reálném čase ze zvukových dat s omezenými hardwarovými možnostmi mobilních zařízení. Přestože se tuto problematiku nikde nepodařilo najít popsanou, tak se ji povedlo vyřešit, avšak tento průzkum zabral více času než se původně předpokládalo.

Ve 4. kapitole byl vytvořen návrh funkčních a nefunkčních požadavků aplikace. Následně byl implementován prototyp aplikace (v 5. kapitole), který obsahuje základní pomůcky používané hudebníky (ladičku a metronom), notové zápisy stupnice a také hru, ve které uživatel musí poznat zobrazenou notu v notovém zápise (zadávat může jak nástrojem, tak bez něj). Tato hra obsahuje všechny (neduplicitní) stupnice, ze kterých si uživatel může sám zvolit, které se budou objevovat ve hře. Pokud tedy chce uživatel zkusit nějakou větší výzvu, může si zvolit jednu z náročnějších stupnic. Pro samotnou motivaci byly na hlavní obrazovce zobrazeny základní statistiky o celkovém a denním získaném skóre.

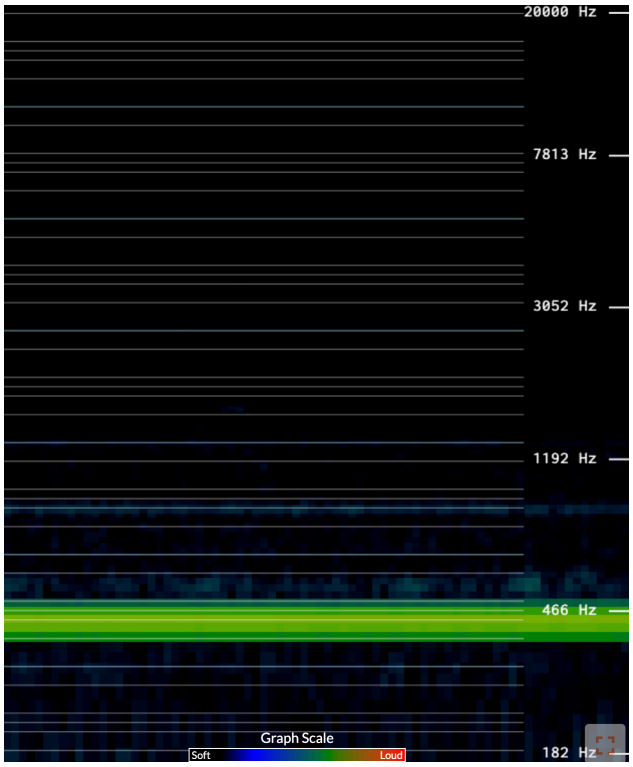
Samotná část aplikace pro zpracovaní zvuku a vykreslení not na platformě Android bude nejspíše zveřejněna pod *Copyleft* licencí, která podpoří vývoj dalších podobně zaměřených aplikací. Zároveň to otevře možnost růstu této knihovny, jelikož další uživatelé této knihovny budou moci přispívat nápady na rozšíření.

Příloha A

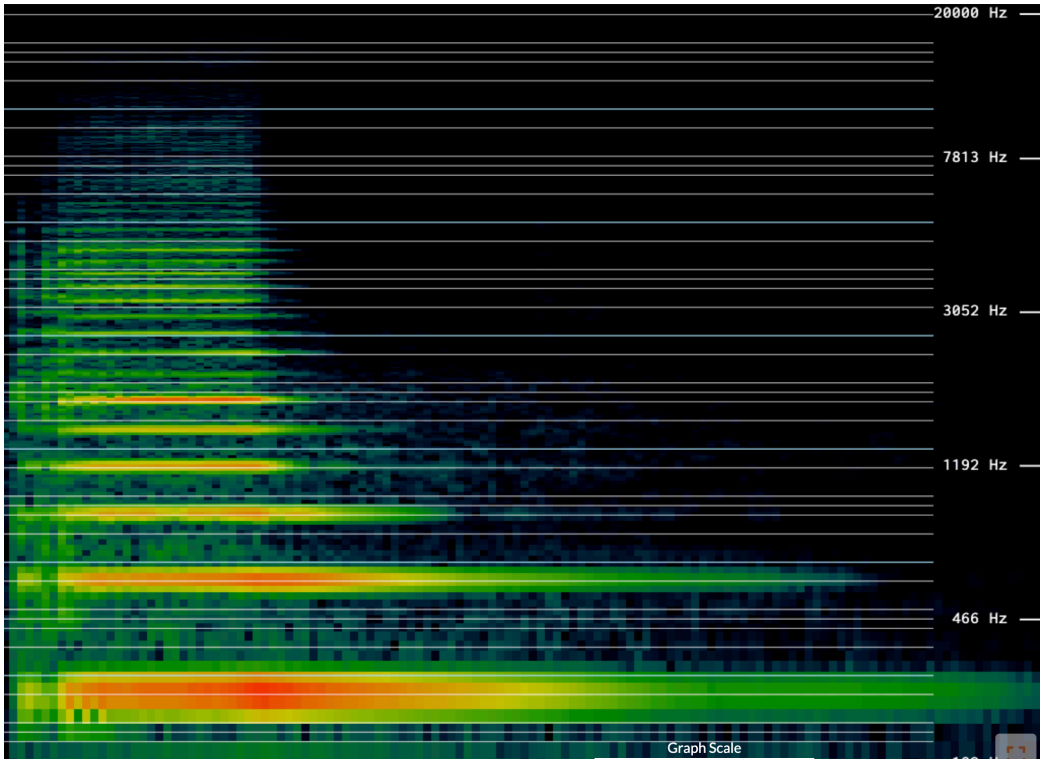
Příloha



Obrázek A.1 Výpis Detektu (vlevo je modul violin a vpravo common)



Obrázek A.2 Zvukové spektrum generátoru (Spektrogram dostupný na [64])



Obrázek A.3 Zvukové spektrum houslí (Spektrogram dostupný na [64])

```
dependencies {
    implementation project(path: ':common')

    def composeBom : Provider<MinimalExternalModuleDependency> = platform(libs.androidx.compose.bom)
    implementation(composeBom)
    androidTestImplementation(composeBom)

    implementation libs.bundles.compose.bom

    implementation libs.compose.destinations
    ksp libs.compose.destinations.ksp

    implementation libs.bundles.di.koin

    implementation libs.logging.timber

    implementation platform(libs.firebase.bom)
    implementation libs.bundles.firebase.bom

    implementation libs.bundles.accopmanist

    implementation libs.bundles.datastore
```

Obrázek A.4 Aplikcace verzovacího katalogu v sestavovacím skriptu

```toml
[versions]
compose-compiler = "1.4.3"
compose-bom = "2023.03.00"
kotlin = "1.8.10"
ksp = "1.8.10-1.0.9" # first part of version is kotlin version
androidStudio = "8.0.0-beta03" # this version depends on android studio version
raamcosta = "1.8.38-beta"
lifeccle = "2.6.1"
koin = "3.2.0"
datastore = "1.1.0"
accompanist = "0.30.0"
room_version = "2.5.1"
junit-jupiter = "5.9.3"

[libraries]
accompanist-systemuicontroller = { group = "com.google.accompanist", name = "accompanist-systemuicontroller",version.ref =
"accompanist" }
accompanist-permissions = { group = "com.google.accompanist", name = "accompanist-permissions", version.ref =
"accompanist" }

androidx-compose-bom = { group = "androidx.compose", name = "compose-bom", version.ref = "compose-
bom" }
androidx-compose-foundation = { group = "androidx.compose.foundation", name = "foundation" }
androidx-core-ktx-bom = { group = "androidx.core", name = "core-ktx" }
androidx-lifecicle-ktx-bom = { group = "androidx.lifecycle", name = "lifecycle-runtime-ktx"}
androidx-lifecycle-extension-bom = { group = "androidx.lifecycle", name = "lifecycle-extensions"}
androidx-lifecycle-viewmodel = { group = "androidx.lifecycle", name = "lifecycle-viewmodel-compose", version.ref = "lifeccle"}
androidx-lifecycle-runtime = { group = "androidx.lifecycle", name = "lifecycle-runtime-compose", version.ref = "lifeccle"}
```

Obrázek A.5 Verzovací katalog TOML

Bibliografie

1. ŠRÁMEK, Filip. *Mobilní technologie ve výuce [online]*. 2017 [cit. 2023-05-04]. Dostupné také z: `https://theses.cz/id/anlyuu/`. Diplomová práce. Univerzita Hradec Králové, Pedagogická fakultaHradec Králové. SUPERVISOR: Mgr. Václav Maněna, Ph.D.
2. *Trala* [online]. Chicago, Illinois: Trala Inc., 2023 [cit. 2023-04-17]. Dostupné z: `https://www.trala.com/`.
3. *Yousician* [online]. Finland: Yousician, 2023 [cit. 2023-04-17]. Dostupné z: `https://yousician.com/`.
4. *Simply* [online]. Israel: JoyTunes Ltd., 2023 [cit. 2023-04-17]. Dostupné z: `https://www.hellosimply.com/`.
5. *ABRSM* [online]. 4 London Wall Place: The Associated Board of the Royal Schools of Music, 2019 [cit. 2023-04-17]. Dostupné z: `https://abrsm.org`.
6. *Perfect Ear - Music Theory, Ear & Rhythm Training* [online]. Sweeden: Crazy Ootka Software AB, 2022 [cit. 2023-04-17]. Dostupné z: `https://www.perfectear.app/`.
7. *MyEarTraining* [online]. Novomeskeho 2, Nitra 94911, Slovakia: MyrApps s.r.o, 2022 [cit. 2023-04-17]. Dostupné z: `https://www.myeartraining.net/`.
8. *Noutee* [online]. Kotlářská 911/31 602 00 Brno: Noutee s r.o., 2019 - 2023 [cit. 2023-04-17]. Dostupné z: `https://noutee.cz/`.
9. FÍBEK, Michal. *Interaktivní e-learningová aplikace pro podporu hudební nauky na ZŠ [online]*. 2015 [cit. 2023-04-17]. Dostupné také z: `https://theses.cz/id/1maiwp/`. Diplomová práce. Mendelova univerzita v Brně, Provozně ekonomická fakultaBrno. SUPERVISOR: Ing. Ondřej Popelka, Ph.D.
10. *Flutter vs Kotlin Multiplatform: The 2023 Guide* [online]. San Francisco, CA: Instabug, Inc., 2023 [cit. 2023-05-01]. Dostupné z: `https://www.instabug.com/blog/flutter-vs-kotlin-mutliplatform-guide`.
11. *Kotlin for Android* [online]. Na Hřebenech II 1718/8, 140 00 Praha 4-Nusle: JetBrains s.r.o., 2023 [cit. 2023-05-01]. Dostupné z: `https://kotlinlang.org/docs/android-overview.html`.
12. PEDRYC, Bartłomiej. *Pros and Cons of Developing an App in Kotlin Multiplatform* [online]. Nowe Garbary Office Center ul. Małe Garbary 9 61-756 Poznań, Poland: Netguru S.A., 2023 [cit. 2023-04-15]. Dostupné z: `https://www.netguru.com/blog/kotlin-multiplatform-pros-and-cons`.
13. *Android Studio: Meet Android Studio* [online]. Mountain View, California 94043, US: Android Developers, 2023 [cit. 2023-05-01]. Dostupné z: `https://kotlinlang.org/docs/android-overview.html`.

14. *Google makes Kotlin a first-class language for writing Android apps* [online]. San Francisco, CA: TechCrunch, 2005 [cit. 2023-04-12]. Dostupné z: https://techcrunch.com/2017/05/17/google-makes-kotlin-a-first-class-language-for-writing-android-apps/.
15. [online]. Na Hřebenech II 1718/8, 140 00 Praha 4-Nusle: JetBrains s.r.o., 2023 [cit. 2023-04-12]. Dostupné z: https://kotlinlang.org/docs/null-safety.html.
16. *Extensions* [online]. Na Hřebenech II 1718/8, 140 00 Praha 4-Nusle: JetBrains s.r.o., 2023 [cit. 2023-05-01]. Dostupné z: https://kotlinlang.org/docs/extensions.html.
17. *Delegation* [online]. Na Hřebenech II 1718/8, 140 00 Praha 4-Nusle: JetBrains s.r.o., 2023 [cit. 2023-05-01]. Dostupné z: https://kotlinlang.org/docs/delegation.html.
18. *Basic Syntax: Variables* [online]. Na Hřebenech II 1718/8, 140 00 Praha 4-Nusle: JetBrains s.r.o., 2023 [cit. 2023-05-01]. Dostupné z: https://kotlinlang.org/docs/basic-syntax.html#variables.
19. *Kotlin language specification: Type inference* [online]. Na Hřebenech II 1718/8, 140 00 Praha 4-Nusle: JetBrains s.r.o., 2023 [cit. 2023-05-01]. Dostupné z: https://kotlinlang.org/spec/type-inference.html.
20. *Kotlin Documentation*. Na Hřebenech II 1718/8, 140 00 Praha 4-Nusle: JetBrains s.r.o., 2023. Dostupné také z: https://kotlinlang.org/docs/home.html.
21. *Coroutine context and dispatchers: Type inference* [online]. Na Hřebenech II 1718/8, 140 00 Praha 4-Nusle: JetBrains s.r.o., 2023 [cit. 2023-05-01]. Dostupné z: https://kotlinlang.org/docs/coroutine-context-and-dispatchers.html#dispatchers-and-threads.
22. *Improve app performance with Kotlin coroutines* [online]. Mountain View, California 94043, US: Android Developers, 2023 [cit. 2023-05-01]. Dostupné z: https://developer.android.com/kotlin/coroutines/coroutines-adv.
23. *Asynchronous Flow* [online]. Na Hřebenech II 1718/8, 140 00 Praha 4-Nusle: JetBrains s.r.o., 2023 [cit. 2023-05-01]. Dostupné z: https://kotlinlang.org/docs/flow.html.
24. *Flow Documentation* [online]. Na Hřebenech II 1718/8, 140 00 Praha 4-Nusle: JetBrains s.r.o., 2023 [cit. 2023-05-01]. Dostupné z: https://kotlinlang.org/api/kotlinx.coroutines/kotlinx-coroutines-core/kotlinx.coroutines.flow/.
25. *StateFlow and SharedFlow* [online]. Mountain View, California 94043, US: Android Developers, 2023 [cit. 2023-05-01]. Dostupné z: https://developer.android.com/kotlin/flow/stateflow-and-sharedflow.
26. *Buffer* [online]. Mountain View, California 94043, US: Android Developers, 2023 [cit. 2023-05-01]. Dostupné z: https://developer.android.com/reference/java/nio/Buffer.
27. *Thinking in Compose* [online]. Mountain View, California 94043, US: Android Developers, 2023 [cit. 2023-05-01]. Dostupné z: https://developer.android.com/jetpack/compose/mental-model.
28. *Follow best practices: Defer reads as long as possible* [online]. Mountain View, California 94043, US: Android Developers, 2023 [cit. 2023-05-01]. Dostupné z: https://developer.android.com/jetpack/compose/performance/bestpractices#defer-reads.
29. *Jetpack Compose Canvas* [online]. 799 Market Street 5th Floor San Francisco, CA 94103 United States of America: Medium, 2012 [cit. 2023-05-01]. Dostupné z: https://medium.com/falabellatechnology/jetpack-compose-canvas-8aee73eab393.
30. *Graphics in Compose* [online]. 799 Market Street 5th Floor San Francisco, CA 94103 United States of America: Medium, 2012 [cit. 2023-05-01]. Dostupné z: https://developer.android.com/jetpack/compose/graphics/draw/overview.
31. *Navigating with Compose* [online]. Mountain View, California 94043, US: Android Developers, 2023 [cit. 2023-05-01]. Dostupné z: https://developer.android.com/jetpack/compose/navigation.

32. *Compose Destinations* [online]. Leiria, Portugal: Rafael Costa, 2023 [cit. 2023-04-13]. Dostupné z: https://composedestinations.rafaelcosta.xyz/.
33. *Jetpack Navigation Compose Animation* [online]. World: Android Cominity, 2023 [cit. 2023-05-02]. Dostupné z: https://google.github.io/accompanist/permissions/.
34. *Material Design* [online]. Mountain View, California, United States: Google, 2023 [cit. 2023-04-12]. Dostupné z: https://m3.material.io/.
35. *Migrate from Material 2 to Material 3 in Compose* [online]. Mountain View, California 94043, US: Android Developers, 2023 [cit. 2023-05-01]. Dostupné z: https://developer.android.com/jetpack/compose/designsystems/material2-material3.
36. *Enable users to personalize their color experience in your app* [online]. Mountain View, California 94043, US: Android Developers, 2023 [cit. 2023-05-01]. Dostupné z: https://developer.android.com/develop/ui/views/theming/dynamic-colors.
37. *Permissions on Android* [online]. Mountain View, California 94043, US: Android Developers, 2023 [cit. 2023-05-01]. Dostupné z: https://developer.android.com/guide/topics/permissions/overview.
38. *Jetpack Compose Permissions* [online]. World: Android Cominity, 2023 [cit. 2023-05-02]. Dostupné z: https://google.github.io/accompanist/permissions/.
39. *Save simple data with SharedPreferences* [online]. Mountain View, California 94043, US: Android Developers, 2023 [cit. 2023-05-01]. Dostupné z: https://developer.android.com/training/data-storage/shared-preferences.
40. *Save data in a local database using Room* [online]. Mountain View, California 94043, US: Android Developers, 2023 [cit. 2023-05-01]. Dostupné z: https://developer.android.com/training/data-storage/room.
41. *Guide to app architecture* [online]. Mountain View, California 94043, US: Android Developers, 2023 [cit. 2023-05-01]. Dostupné z: https://developer.android.com/topic/architecture.
42. SEEMANN, Mark. *Dependency Injection is Loose Coupling* [online]. Dánsko: Ploeh blog, 2009 - 2023 [cit. 2023-04-16]. Dostupné z: https://blog.ploeh.dk/2010/04/07/DependencyInjectionisLooseCoupling/.
43. *Koin* [online]. France: Koin & Kotzilla, 2023 [cit. 2023-04-20]. Dostupné z: https://insert-koin.io/.
44. *Hilt* [online]. Mountain View, California, United States: Google, 2023 [cit. 2023-04-20]. Dostupné z: https://dagger.dev/hilt/.
45. *UI layer* [online]. Mountain View, California 94043, US: Android Developers, 2023 [cit. 2023-05-01]. Dostupné z: https://developer.android.com/topic/architecture/ui-layer.
46. IAN, Alexander. *Architecture in Jetpack Compose — MVP, MVVM, & MVI* [online]. 799 Market Street 5th Floor San Francisco, CA 94103 United States of America: Medium, 2012 [cit. 2023-04-16]. Dostupné z: https://medium.com/mobile-at-octopus-energy/architecture-in-jetpack-compose-mvp-mvvm-mvi-17d8170a13fd.
47. *Guide to Android app modularization* [online]. Mountain View, California 94043, US: Android Developers, 2023 [cit. 2023-05-01]. Dostupné z: https://developer.android.com/topic/modularization.
48. *Configure your build* [online]. Mountain View, California 94043, US: Android Developers, 2023 [cit. 2023-05-01]. Dostupné z: https://developer.android.com/build.
49. *Using the Bill of Materials* [online]. Mountain View, California 94043, US: Android Developers, 2023 [cit. 2023-05-01]. Dostupné z: https://developer.android.com/jetpack/compose/bom.

50. *Sharing dependency versions between projects* [online]. San Francisco, CA: Gradle Inc, 2022 [cit. 2023-05-02]. Dostupné z: https://docs.gradle.org/current/userguide/platforms.html.

51. *Migrate your build to version catalogs* [online]. Mountain View, California 94043, US: Android Developers, 2023 [cit. 2023-05-02]. Dostupné z: https://developer.android.com/build/migrate-to-catalogs.

52. BLACK, H. S.; EDSON, J. O. Pulse code modulation. *Transactions of the American Institute of Electrical Engineers*. 1947, roč. 66, č. 1, s. 895–899. Dostupné z DOI: 10.1109/T-AIEE.1947.5059525.

53. *AudioRecord* [online]. Mountain View, California 94043, US: Android Developers, 2023 [cit. 2023-05-02]. Dostupné z: https://developer.android.com/reference/android/media/AudioRecord.

54. KUTTNER, Fritz A. Prince Chu Tsai-Yü's Life and Work: A Re-Evaluation of His Contribution to Equal Temperament Theory. *Ethnomusicology* [online]. 1975, roč. 19, č. 2, s. 163–206 [cit. 2023-04-12]. ISSN 00141836. Dostupné z: http://www.jstor.org/stable/850355.

55. *Notes* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2023-04-22]. Dostupné z: https://commons.wikimedia.org/wiki/File:Notes.svg.

56. *Digital Signals Theory* [online]. New York: New York University, 2020-2022 [cit. 2023-04-12]. Dostupné z: https://brianmcfee.net/dstbook-site/content/intro.html.

57. *Euler's formula* [online]. San Francisco (CA): Wikimedia Foundation, 2001-2023 [cit. 2023-04-18]. Dostupné z: https://commons.wikimedia.org/wiki/File:Euler%27s_formula.png.

58. COCHRAN, W.T.; COOLEY, J.W.; FAVIN, D.L.; HELMS, H.D.; KAENEL, R.A.; LANG, W.W.; MALING, G.C.; NELSON, D.E.; RADER, C.M.; WELCH, P.D. What is the fast Fourier transform? *Proceedings of the IEEE*. 1967, roč. 55, č. 10, s. 1664–1674. Dostupné z DOI: 10.1109/PROC.1967.5957.

59. HEIDEMAN, Michael; JOHNSON, Don; BURRUS, Charles. Gauss and the history of the fast Fourier transform. *IEEE Assp Magazine*. 1984, roč. 1, č. 4, s. 14–21.

60. *Harmonic Product Spectrum (HPS)* [online]. UC San Diego 9500 Gilman Dr. La Jolla: University of California, 2023 [cit. 2023-04-26]. Dostupné z: http://musicweb.ucsd.edu/~trsmyth/analysis/Harmonic_Product_Spectrum.html.

61. *Harmonic Product Spectrum Diagram* [online]. United Kingdom: UK Research a Inovation, 2023 [cit. 2023-04-26]. Dostupné z: http://www.mazurka.org.uk/software/sv/plugin/MzHarmonicSpectrum/img/hpsdiagram.png.

62. PLONKA, Gerlind; POTTS, Daniel; STEIDL, Gabriele; TASCHE, Manfred. *Numerical Fourier Analysis*. Springer, 2018. ISBN 978-3-030-04305-6.

63. *Frequency Sound Generator* [online]. Google Ireland Limited, Gordon House, Barrow Street, Dublin 4, Ireland: Google Play, 2021-03-12 [cit. 2023-05-03]. Dostupné z: https://play.google.com/store/apps/details?id=com.boedec.hoel.frequencygenerator&hl=en&gl=US.

64. *Spectogram* [online]. San Diego, Southern California: University of California, 2016 [cit. 2023-05-03]. Dostupné z: https://spectrogram.sciencemusic.org/.

Obsah přiloženého média

- `Zdrojové kódy`
  - `MusicApps` ..... zdrojové kódy implementace
    - `common` ..... zdrojové kódy společné části
    - `violin` ..... zdrojové kódy části pro housle
    - `LICENCE` ..... Licence pro použití zdrojových kódů
  - `README.md` ..... stručný popis hlavních částí v projektu
- `Thesis`
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
  - `thesis.pdf` ..... text práce ve formátu PDF