VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

MOBILNÍ APLIKACE PRO VÝUKU HRY NA BICÍ

MOBILE APPLICATION FOR LEARNING TO PLAY THE DRUMS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **JAN NOVOTNÝ**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. VÍTĚZSLAV BERAN, Ph.D.**
SUPERVISOR

BRNO 2021

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav počítačové grafiky a multimédií (UPGM) Akademický rok 2020/2021

Zadání bakalářské práce

Student: **Novotný Jan**
Program: Informační technologie
Název: **Mobilní aplikace pro výuku hry na bicí**
Mobile Application for Learning to Play Drums
Kategorie: Uživatelská rozhraní

Zadání:

1. Seznamte se formátem MIDI a nástroji pro jeho vizualizaci a přehrávání na mobilních platformách. Seznamte se s metodami pro zpracování signálů.
2. Navrhněte aplikaci, která bude zobrazovat notaci pro bicí nástroje, přehrávat tento záznam, zaznamenávat zvuk hry na bicí nástroj a tento záznam analyzovat s cílem vizualizace nepřesností a chyb.
3. Implementujte aplikaci pro vybranou mobilní platformu s využitím vhodných knihoven.
4. Vyhodnoťte vlastnosti výsledné aplikace na základě experimentů v reálném prostředí.
5. Prezentujte klíčové vlastnosti řešení formou plakátu a krátkého videa.

Literatura:

- Joel Marsh. *UX pro začátečníky.* Zoner Press, 2019.
- Jan, J., Kozumplík, J. *Systémy, procesy a signály.* Skriptum VUT v Brně, VUTIUM, 2000.
- Pavel Tišnovský. General MIDI a formát souborů SMF. Root.cz. 2009. Url: https://www.root.cz/clanky/general-midi-a-format-souboru-smf/
- Dále dle pokynu vedoucího.

Pro udělení zápočtu za první semestr je požadováno:

- Body 1, 2 a částečně bod 3.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Beran Vítězslav, Ing., Ph.D.**
Vedoucí ústavu: Černocký Jan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2020
Datum odevzdání: 12. května 2021
Datum schválení: 30. října 2020

Abstrakt

Cílem této bakalářské práce je navrhnout a vytvořit mobilní aplikaci pro výuku hry na bicí nástroje a trénování rytmického čtení. Aplikace obsahuje několik lekcí se cvičeními, které jsou zobrazované v podobě notového zápisu. Je umožněno přehrávání a procvičování jednotlivých cvičení s doprovodem metronomu. Dále nabízí analýzu hry na nástroj a poskytuje zpětnou vazbu o kvalitě hraní. Aplikace je vyvíjená na platformě Xamarin pro operační systémy Android a iOS. Výsledkem práce je aplikace, která poslouží bubeníkům k procvičování s hodnocením přesnosti odehraného cvičení a tím napomůže ke zdokonalování jejich hracích schopností.

Abstract

This bachelor thesis aims to design and create a mobile application for teaching percussion playing and rhythmic reading training. The application contains several lessons with exercises that are displayed in the form of music notation. It is made possible to play and practice individual exercises accompanied by a metronome. Furthermore, it offers an analysis of the play on the instrument and provides feedback on the quality of playing. The application is developed on the Xamarin platform for use on Android and iOS operating systems. The result of this work is an application that will serve drummers to practice and provide an evaluation of the accuracy of the played exercise and thus will help to improve their playing skills.

Klíčová slova

Mobilní aplikace, bicí nástoje, Xamarin, Android, detekce úderů, zpracování signálů, učení se zpětnou vazbou

Keywords

Mobile application, drums, percussion, Xamarin, Android, hit detection, signal processing, learning with feedback

Citace

NOVOTNÝ, Jan. *Mobilní aplikace pro výuku hry na bicí*. Brno, 2021. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Vítězslav Beran, Ph.D.

Mobilní aplikace pro výuku hry na bicí

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Vítězslava Berana Ph.D. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Jan Novotný
10. května 2021

Poděkování

Velmi rád bych poděkoval vedoucímu práce panu Ing. Vítězslavu Beranovi Ph.D. za jeho odborné vedení, cenné rady a trpělivost při tvorbě této práce. Dále bych chtěl poděkovat rodině a přátelům za morální podporu a všem, kteří se podíleli na testování.

Obsah

Kapitola 1

Úvod

V dnešní době, kdy je dostupnost hudebních nástrojů velmi vysoká, láká dovednost na některý z nich hrát stále více lidí. Bicí jsou vedle kytary a klavíru nejpopulárnějším hudebním nástrojem. Naučit se hrát na bicí nástroje je však složitý a časově náročný proces. Nejlepší způsob, jak se naučit hrát na bicí nebo jakýkoli jiný hudební nástroj, je učení se zkušeným odborníkem. Mezi základy hry na bicí patří takzvané rudimenty, což jsou základní vzory, ze kterých se sestavují složitější rytmy. Další základní dovedností bubeníka je čtení notového zápisu a schopnost tento zápis zahrát. Cvičení základních technik a rytmů je důležité v jakékoliv fázi výuky – jak pro začátečníky, tak pro pokročilé. V době chytrých telefonů není problém mít noty ke cvičení vždy u sebe a vyrazit cvičit s paličkami a tréninkovým padem třeba do parku. Samostudium ale často vede k utvoření špatných návyků, kterých se těžko zbavuje.

Cílem této práce je navrhnout a vytvořit mobilní aplikaci sloužící jako pomocník k výuce a procvičování hry na bicí nástroje. Aplikace má sloužit jako podpora pro cvičení na vířivý buben nebo tréninkový pad se zaměřením na rytmické čtení. Bude obsahovat několik lekcí s vysvětlením základních symbolů zápisu notace, odstupňovaných podle úrovně náročnosti. Každé z nich bude zobrazovat notový zápis a umožňovat uživateli nastavit si tempo, v jakém chce cvičení přehrávat. Důležitým prvkem této aplikace je právě eliminace tvoření si špatných návyků tím, že má nahradit dozor odborníka. Aplikace proto bude poskytovat zpětnou vazbu uživateli o jeho hraní a zobrazovat mu výsledky. Při přehrávání cvičení na tréninkový pad nebo vířivý buben bude aplikace zaznamenávat hru pomocí mikrofonu a záznam pak analyzovat s cílem vyhledání zahraných úderů.

Tato práce se skládá z několika částí. V teoretické části se zabývá přiblížením teoretického základu, který poslouží k lepší pochopení problematiky hry na bicí nástroje, mobilních platforem a základům zpracování signálů. V následující části je popsána problematika, kterou tato práce řeší, dekompozice problému a návrh aplikace. V implementační části je popsána architektura aplikace a způsob její implementace a na závěr je popsáno testování.

Kapitola 2

Teorie a průzkum existujících řešení

Tako kapitola obsahuje teoretický základ, potřebný k porozumění této práce. Na začátku vymezuje pár pojmů týkajících se hry na bicí nástroje. Pak pojednává o existujících řešeních. Věnuje se přiblížení rozhraní MIDI a základům zpracování audio signálů. Nakonec rozebírá a porovnává mobilní platformy a technologie používané pro vývoj mobilních aplikací.

2.1 Hra na bicí nástroje

Jelikož se tato práce zabývá tvorbou aplikace pro výuku bicích, je vhodné jim věnovat jednu podkapitolu. Slouží pro přiblížení cílové skupiny a vymezení pojmů týkajících se bicích nástrojů použitých v této práci.

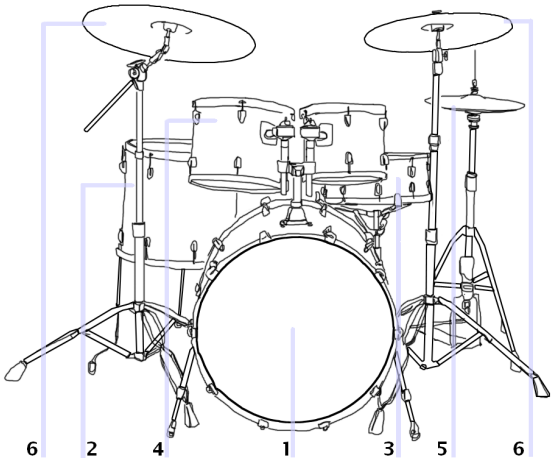
Jednotlivé definice bicích nístrojů byly převzaty z portálu Drum Center a Zing instruments [17][5].

Bicí nástroje (perkuse) jsou nejstarší a nejrozsáhlejší skupinou hudebních nástrojů. K vytváření zvuku dochází pomocí úderu do nástroje, třesením, škrábáním nebo úderem o jiný podobný nástroj. Pojem perkuse zahrnuje všechny bicí nástroje, avšak v moderní době došlo k trochu nepřesnému rozdělení pojmu na perkuse a bicí soupravu.

Bicí souprava, znázorněna na obrázku 2.1, je označení pro sestavu několika bicích nástrojů do jedné sady tak, aby bylo možné hrát na všechny najedou. Skládá se z jednotlivých bubnů, činelů, jejich stojanů, přechodů a dalších perkusních nástrojů. Velký buben a hi-hat činel je ovládán šlapkou nebo dvojšlapkou.

Vířivý buben, také označovaný jako malý buben nebo Snare, je mělký buben umístěný přímo před bubeníkem. Hraje se na něj rukou, v níž je uchopena palička. Součástí je struník, což je několik pružin umístěných na spodní bláně. Díky svým rozměrům a struníku má charakteristický vysoký zvuk.

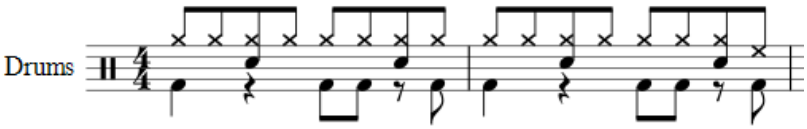
Tréninkový pad je malá, snadno přenosná pomůcka bubeníka, určená k procvičování, která je oproti bicím při hraní tichá. Jedná se o plastovou nebo dřevěnou desku, z vrchu pokrytou gumou, na kterou se hraje. Ta má odrazové vlastnosti podobné bláně vířivého bubnu. Slouží pro trénování techniky rukou.



Obrázek 2.1: Bicí souprava, 1 – Basový buben, 2 – Floor tom, 3 – vířivý buben, 4 – Přechody (Tom tomy), 5- Hi-Hat činel, 6 – Crash a Ride činely. (Převzato z: [1].)

Nota je základní grafický symbol pro označení tónu. K zápisu not slouží notová osnova. Tvar noty určuje její hodnotu, která udává relativní délku trvání tónu. Umístění noty v notové osnově obvykle určuje výšku tónu, ale u zápisu pro bicí soupravu se jedná o označení konkrétního bubnu [7].

Cvičení – v kontextu této práce se cvičením rozumí skladba určená pro procvičování hry na bicí a vyjádřená v notovém zápise. Notový zápis (notace) na obrázku 2.2 je způsob záznamu hudební skladby pomocí not.



Obrázek 2.2: Notace pro bicí soupravu

Rudimenty – jedná se o krátké rytmy, které slouží jednak pro procvičování techniky rukou, ale také jako vzory pro skládání složitějších rytmů.

Tempo, vyjádřené v **BPM** (beats per minute) určuje v hudbě rychlost pohybu v čase. Každá skladba má své tempo, které se v notovém zápisu vyjadřuje počtem čtvrťových not za minutu.

2.2 Existující aplikace

Bubeník potřebuje trénovat různé techniky a rudimenty. To je důležité pro procvičování motoriky rukou, rychlosti, správného držení rytmu atd. Dále k tomu patří schopnost číst notový zápis. Existuje množství učebnic a příruček určených na procvičování rytmů na vířivý buben a rytmického čtení.

V dnešní mobilní době je trhu spousta aplikací, které mají sloužit jako pomůcka k výuce. Při zkoumání existujících aplikací bylo vyzkoušeno několik aplikací určených pro bubeníky

na platformě Android. Bubenické aplikace, které jsou k dispozici, se dají rozdělit do čtyř základních skupin:

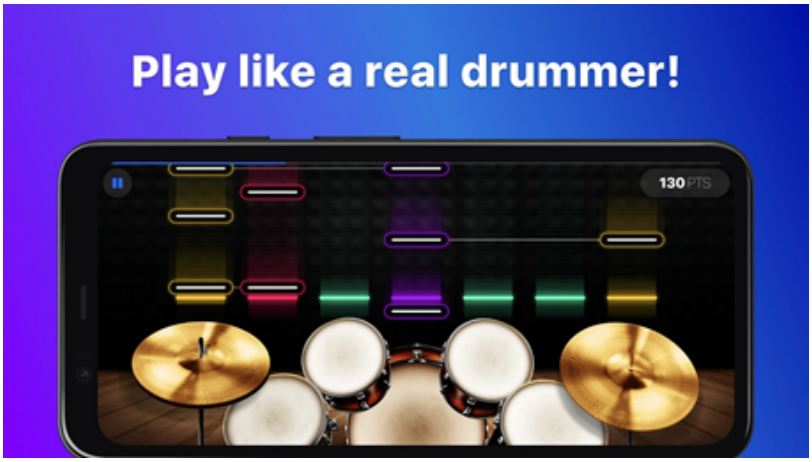
Simulátory bicích nátrojů

Jedná se o aplikace, které zobrazí bicí soupravu a uživatel může klikat na jednotlivé bubny, které zahrají příslušný zvuk. U některých aplikací lze soupravu modifikovat. Tyto aplikace jsou určené hlavně pro pobavení, rychle se omrzí a s výukou nemají nic společného. Mezi tyto aplikace patří třeba:

- REAL DRUM: Electronic Drum Set[1]
- Simple Drums Rock - Realistic Drum Simulator[2]
- Snare Drum Pro[3]

Aplikace typu Guitar Hero

Je to v podstatě *Guitar Hero*[4], jen s bicí soupravou (obrázek 2.3). V pozadí hraje hudba a do rytmu jezdí barené tvary k příslušnému bubnu, uživatel musí včas na buben kliknout. Jedná se o hru, ne o výukový materiál.



Obrázek 2.3: Aplikace Drums: real drum set music games to play and learn (Převzato z: [2].

Mezi tyto aplikace patří třeba:

- Drums: real drum set music games to play and learn[5].
- Drum Set Music Games & Drums Kit Simulator[6]

[1] https://play.google.com/store/apps/details?id=br.com.rodrigokolb.realdrum
[2] https://play.google.com/store/apps/details?id=com.tpvapps.simpledrumsrock
[3] https://play.google.com/store/apps/details?id=com.qpstudios.snaredrumpro
[4] https://www.guitarhero.com
[5] https://play.google.com/store/apps/details?id=com.mwm.drum
[6] https://play.google.com/store/apps/details?id=com.gismart.realdrum2free

Výukové podpůrné aplikace

Skupina, ve které se dají najít i dobré aplikace, které mohou sloužit jako podpora pro trénování. Tyto aplikace třeba zobrazují notový zápis rudimentů nebo rytmů na bicí soupravu, které umí přehrát. Dále bych sem zařadil spoustu podpůrných aplikací nebo aplikace sloužící pro inspiraci. Například na ukazování různých rytmů nebo přímo jejich generování.
Mezi tyto aplikace patří:

- **Drumate - Drum Rudiments** [7] – Aplikace na procvičování rudimentů. Zobrazuje noty rudimentů a dokáže je přehrát ve zvoleném tempu.
- **Loopz - Best Drum Loops!**[8] – Aplikace přehrávající rytmy na bicí soupravu. Obsahuje rytmy různých žánrů a taktů.
- **Linear Drum Patterns Generator**[9] – Vygeneruje jeden takt náhodného rytmu.
- **Drum Sheets Reading**[10] – Trochu podobná Drum Simulátoru s tím, že používá klasický notový zápis.
- **Snare drum study**[11] – Učí noty a držení paliček. Jednotlivé cvičení přehraje.

Výukové video aplikace

Patří sem aplikace pro přehrávání video návodů, jak držet paličky, šlapat na pedál a hrát základní rytmy na bicí soupravu. Mezi těmito aplikacemi se dají najít skutečně dobré a užitečné aplikace.

Jednou z těchto aplikací je **Drumeo**[12]. To je jedna z nejlepších, ne-li nejlepší výuková aplikace vůbec. Drumeo je portál s obrovským množstvím lekcí od profesionálních bubeníků z celého světa. Pro přístup k lekcí je třeba zaplatit členství, ale poskytuje spoustu lekcí na youtube zdarma.

2.3 Rozhraní MIDI

Součástí této práce je tvorba aplikace, která zpracovává MIDI soubory. K tomu je potřeba toto rozhraní nastudovat a pochopit, jak jsou jednotlivá data reprezentována. Zde je popsáno toto rozhraní a formát souboru SMF.

Musical Instrument Digital Interface (MIDI) je standard sloužící k propojení velkého množství především hudebních nástrojů. MIDI standard je rozdělen na dvě části. První část popisuje hardware, jakým způsobem jsou zařízení propojena, tj. konektory, elektrické charakteristiky, připojení a způsob přenosu dat. Druhá část popisuje přenosový protokol – způsob řízení jednotlivých zařízení.

Přenos dat spočívá v zasílání tzv. MIDI zpráv. MIDI zpráva se skládá z jednoho, dvou nebo nejčastěji tří bytů. První bajt je stavový bajt (status byte), další bajty obsahují dodatečné informace podle typu zprávy [19].

[7] https://play.google.com/store/apps/details?id=com.deltaphi.drumate
[8] https://play.google.com/store/apps/details?id=com.envelopedevelopment.loopz
[9] https://play.google.com/store/apps/details?id=com.vk.patterngenerator
[10] https://play.google.com/store/apps/details?id=air.com.musycom.LeerMusicaEnBateria
[11] https://play.google.com/store/apps/details?id=com.yarolegovich.discretescrollview.snaredrum
[12] https://www.drumeo.com

Status byte

V prvním bajtu je uložen kód typu zprávy a číslo kanálu. Kanál je v podstatě adresa zařízení, která nabývá hodnot 0–15. První bit je nastaven vždy na 1, proto lze efektivně využít jen 7 bitů. 4 bity jsou využité pro číslo kanálu a zbývající 3 obsahují kód typu zprávy. Jejich výčet je v tabulce 2.1.

Kód příkazu (hex)	**Název**
0x80	Note Off
0x90	Note On
0xa0	Aftertouch
0xb0	Continuous Controller
0xc0	Path Change
0xd0	Chanell Pressure
0xe0	Pitch bend
0xf0	(non-musical commands

Tabulka 2.1: Výčet MIDI událostí

Všechny MIDI zařízení by měly reagovat alespoň na zprávy Note On (přehrávání tónu) a Note Off (ukončení přehrávání tónu). Obě zprávy mají délku tři bajty. První bajt je stavový. Ve druhém bajtu je uložen kód noty a ve třetím rychlost přehraného tónu.

General MIDI (GM) rozšiřuje a upravuje normu MIDI. Zařízení GM-compatible musí dokázat přehrát minimálně 24 tónů zároveň, musí podporovat všech 16 kanálů, přičemž kanál 10 je vyhrazen pro bicí nástroje, definuje čísla nástrojů rozdělených do skupin [19].

Formát souborů SMF

V GM je specifikován i formát souborů SMF (Standard MIDI file). Jedná se o binární soubory s příponou (nejčastěji) .mid. Jedná se mezi hudebníky o velice rozšířený formát, a to především díky podpoře v hudebních nástrojích a v počítačích a úspornému záznamu hudby. V SMF souborech může být uložena buď jedna stopa, anebo (v případě více nástrojů) stop více. Oproti jiným hudebním formátům zde nejsou uložena zvuková data, ale jen informace o použitých nástrojích, tempu, kanálech a posloupnost jednotlivých událostí, což jsou v podstatě MIDI zprávy doplněny o časovou značku. Vlastní zvuky hudebních nástrojů v souborech SMF uloženy nejsou [18].

Data jsou v souboru uložena blocích (chunk). Ty můžou být dvou typů:

- **MThd** – Hlavička souboru
- **MTrk** – Data a parametry stopy

Každý chunk obsahuje čtyřbajtový textový identifikátor a informaci o délce chunku (4 bajty). Hlavička pak navíc obsahuje informace o typu SMF souboru, počtu stop a rychlosti přehrávání.

Data Stopy

Kromě identifikátoru a délky obsahují tzv. události (events a meta-events). Jsou to MIDI zprávy s časovou značkou. Nejjednodušší událost má 4 bajty, přičemž v prvním bajtu je uložena časová značka, po níž následuje klasická MIDI zpráva.

Bajty v souboru	Událost	Význam
00 90 3C 60	Note On	Začne se přehrávat nota C
7F 90 3E 60	Note On	Začne se přehrávat nota D
7F 90 40 60	Note On	Začne se přehrávat nota E
7F 80 3C 00	Note Off	Konec přehrávání noty C
00 80 3E 00	Note Off	Konec přehrávání noty D
00 80 40 00	Note Off	Konec přehrávání noty E

Tabulka 2.2: Ukázka souboru SMF

Každá událost začíná časovou značkou. Ta reprezentuje časový rozdíl mezi vykonáním poslední události a událostí, která právě začíná. Jako jednotky času jsou využity tiky, což je relativní časový údaj. V hlavičce MIDI souboru je uložena hodnota BPM a počet tiků na jednu čtvrťovou notu. Tabulka 2.2 ukazuje příklad obsahu stopy souboru SMF.

2.4 Zpracování audio signálů

Důležitou součástí výsledné aplikace je práce s audiozáznamem a jeho analýzou. Tato podkapitola se bude věnovat stručnému, teoretickému základu zpracování signálů, který vychází z práce Martina Mlčocha [15].

Signálem se v technice rozumí stav fyzikální veličiny závislé na čase. Zpracováním signálů je pak proces, kterým jsou ze signálu získávány informace. Pro akustické signály je potřeba je před dalším zpracováním převést z analogové do digitální podoby. Digitální signál lze snadno zpracovávat pomocí výpočetní techniky. U signálu lze získávat souhrnné charakteristiky nebo frekvenční spektra. Mezi základní charakteristiky signálu patří např. energie, střední hodnota, rozptyl a výkon. Frekvenční spektrum představuje podíl frekvencí neboli harmonických složek v signálu.

Digitální signál je vzorkovaný a následně kvantovaný signál. Tedy signál, který je tvořen posloupností vzorků, kde hodnota vzorku nemá spojitý průběh, ale mění se skokem, přičemž nabývá omezeného počtu úrovní. Proto jej lze reprezentovat posloupností celých čísel [20][16].

$$x\left[n\right], O < n < N, n \in Z \tag{2.1}$$

Kde x – signál, n – hodnota vzorku, N – maximální hodnota vzorku

Filtry

Filtrace je druh zpracování signálu, při kterém jsou potlačovány určité složky, a tím získáno požadované spektrum pro další zpracování. Cílem je oddělit užitečnou a rušivou složku. Používá se například k odstranění šumu nebo výběru pouze požadované části spektra.

K propuštění jedné, požadované, části spektra slouží frekvenčně selektivní filtry:

- **Dolní propust** – propouští spodní část spektra
- **Horní propust** – propouští horní část spektra
- **Pásmová propust** – propouští signály pouze v určitém intervalu frekvencí
- **Pásmová zádrž** – blokuje signály v určitém intervalu frekvencí

Dále lze filtry dělit na analogové, realizované pomocí elektronických součástek (rezistory, kondenzátory atd.), a číslicové, využívající pro zpracování signálu procesor, který provádí výpočty se vzorky. Těmi jsou:

- **Filtr s konečnou impulsní odezvou (FIR)** – Na výstupu průchodu impulsu filtrem je konečný počet hodnot a je vždy stabilní. Tento filtr se používá například pro odstranění šumu.
- **Filtr s nekonečnou impulsní odezvou (IIR)** – vyznačuje se zpětnou vazbou, není u nich zaručena stabilita.

Fourierova transformace

Fourierova transformace slouží k převodu signálu z časové oblasti do oblasti frekvenční. Výsledkem tohoto procesu je frekvenční spektrum. Spektrum je rozklad signálu na jednotlivé frekvenční složky. Zjednodušeně řečeno ukazuje, jak moc je která frekvence v signálu zastoupena. Rozlišujeme spojitou Fourierovu transformaci, která pracuje se spojitým signálem a disktrétní pro práci s diskrétními hodnotami signálu.

Parametry signálu pro audio analýzu

Jednou z nich je rámcování. Jelikož je akustický signál považovaný za náhodný, je dobré si ho rozdělit na krátké časové úseky a analýzu provádět nad každým tímto rámcem. Mezi základní parametry patří:

- **Energie signálu** – parametr sloužící ke zjištění energie v určitém intervalu

$$E \;=\; \sum_{n_1}^{n_2} \left(x\left[n\right]\right)^2 \tag{2.2}$$

  Kde $x\left[n\right]$ – vzorek signálu, n_1 – spodní hodnota intervalu, n_2 – horní hodnota intervalu

- **Střední hodnota** – vyjadřuje průměrnou hodnotu po dobu trvaní intervalu.

$$\bar{x} \;=\; \frac{1}{n_1 - n_2 + 1} \sum_{n_1}^{n_2} x\left[n\right] \tag{2.3}$$

  Kde $x\left[n\right]$ – vzorek signálu, n_1 – spodní hodnota intervalu, n_2 – horní hodnota intervalu

- **Rozptyl** – určuje podíl střídavé složky signálu

$$V = \frac{1}{N} \sum_{n=0}^{N} (x_n - \bar{x})^2 \tag{2.4}$$

  Kde x_n jsou hodnoty signálu a $\bar{x}$ je střední hodnota.

2.5 Vývoj mobilních aplikací

Protože je cílem této práce naprogramovat mobilní aplikaci, v této podkapitole boudou stručně představeny mobilní platformy, způsoby vývoje aplikací na tyto platformy a srovnání dostupných technologií. To je důležité pro pozdější volbu použité technologie pro vývoj výsledné aplikace.

Mobilní platformy

Pojem mobilní platforma se užívá v souvislosti s chytrými mobilními telefony, tablety a dalšími přenosnými zařízeními. Jedná se o software, který běží na mobilním zařízení. Různé platformy mají různé operačními systémy, aplikační rozhraní pro vývojáře a disponují různými aplikacemi, pro které jsou pro ni dostupné.

Nejpoužívanějšími mobilními platformami jsou dnes operační systém Android od společnosti Google, které si připisuje 74,45% světového mobilního trhu (k prosinci 2020), a operační systém iOS od společnosti Apple s podílem 22,85% mobilního trhu. Zbytek připadá na ostatní mobilní platformy jako je například Windows Phone, BlackBerry nebo KaiOS [8].

Android

Android je operační systém, který vyvinula společnost Android, Inc. Tu v roce 2005 koupila společnost Google, která na vývoji tohoto operačního systému nadále pokračuje. Je založený na jádře Linuxu a je to open-source software. Jako platforma není používaný jen pro mobilní telefony, ale i pro tablety, chytré hodinky, televize a další zařízení. Výrobci zařízení mohou systém Android upravovat a na svých zařízeních používat různé nadstavby a modifikace. Existuje tedy opravdu široká nabídka zařízení různých značek a specifikací.

Android poskytuje vývojářům rozsáhlé aplikační rozhraní, Android Studio a sadu nástrojů Android Software Development Kit, který je dostupný pro všechny platformy operačních systémů Linux, Windows, macOS, pro vývoj mobilních aplikací pro tuto platformu. Jako primární programovací jazyky sloužící pro vývoj Android aplikací se používají Java[13] nebo Kotlin[14].

iOS

iOS je operační systém od společnosti Apple. Na rozdíl od Androidu není určený pro ostatní výrobce a lze jej instalovat pouze na zařízení Apple. iOS je uzavřený systém. Nelze do něj instalovat jiné aplikace než ty schválené společností Apple, ani jej přizpůsobit pomocí

[13] https://www.java.com
[14] https://kotlinlang.org

nástaveb. To poskytuje vyšší bezpečnost, rychlejší nasazení aktualizací a lepší optimalizaci samotného operačního systému.

Pro vývoj aplikací na tuto platformu se používá programovací jazyk Swift[15], který nahradil dlouho používaný Objective-C.

Multiplatformní vývoj

Mobilní aplikace jsou součástí života většiny lidí. Pokud někdo chce vyvíjet aplikaci pro co největší množství uživatelů, je potřeba, aby zvolil vývoj na více platformách. V dnešní době to znamená zejména Android a iOS. Aplikace se ale musí udržovat, a to při vývoji na tyto dvě platformy vyžaduje značné množství času, neboť se aplikace pro každou platformu vyvíjí zvlášť. Z tohoto důvodu vznikají různé multiplatformní nástroje, které umožňují, sdílením velké části kódu, vývoj pro více platforem najednou. Tím se ušetří čas, a tím pádem i náklady na vývoj.

Vývoj mobilních aplikací lze rozdělit do několika kategorií. Jedná se o nativní vývoj, progresivní webové aplikace, hybridní aplikace a multiplatformní nativní aplikace (Cross-platform Applications).

Nativní

Jedná se o způsob tvorby mobilních aplikací pro specifickou platformu. Je tedy opakem multiplatformního vývoje. Každá platforma využívá svůj programovací jazyk a vývojové prostředí, které poskytuje tvůrce platformy. Vývojáři tak mají přístup k nejnovějšímu aplikačnímu rozhraní, což je vedle vysokého výkonu aplikace nespornou výhodou.

Progresivní webové aplikace

Progresivní webová aplikace (PWA) je aplikace, která vypadá a načítá se stejně jako webové stránky a považují se za skutečné webové aplikace. Pro jejich vývoj jsou použity technologie HTML, CSS, JavaScript. Oproti nim využívají moderní technologie webového prohlížeče HTML5, které umožnuje používat například notifikace a přístup k hardwaru zařízení.

Hybridní

Hybridní aplikace jsou kombinací nativního a webového přístupu. Jedná se v podstatě o webové aplikace, kde pro zobrazení této stránky je vytvořena nativní aplikace obsahující webový prohlížeč pro danou platformu. Vývojář tak tvoří webovou aplikaci, kterou může sdílet mezi všemi platformami. Aplikaci pak může distribuovat na každou platformu zvlášť. Výhody tohoto způsobu vývoje jsou rychlost vývoje, sdílený kód a konzistentní vzhled.

Multiplatformní nativní

Tyto aplikace mají nejblíže k nativním aplikacím. Používá se jeden programovací jazyk, ve kterém se napíše sdílený kód. Ten je pak přeložen pro všechny cílové platformy. Výsledkem je tady pro každou platformu nativní aplikace. Výkonem se tedy blíží nativním aplikacím.

Jako nástroje pro vývoj multiplatformních aplikací se v dnešní době využívají především Xamarin, Flutter nebo React Native.

[15] https://developer.apple.com/swift/

Frameworky pro multiplatformní vývoj aplikací

Pro vývoj multiplatformních aplikací je třeba zvolit vhodnou technologii pro vývoj. Dále budou zmíněné technologie stručně popsány a uvedeny jejich výhody a nevýhody.

Xamarin

Xamarin je open-source platforma spravovaná Microsoftem pro vytváření moderních a výkonných aplikací pro iOS, Android, Windows, Mac, Tizen a další pomocí jazyka C#, .NET a XAML. Pomocí Xamarinu lze vyvíjet aplikace specifické pro platformu nebo pro všechny platformy najednou. K tomu slouží Xamarin.Forms, díky kterému, kromě kódu logiky, je možné sdílet i kód pro uživatelské rozhraní. Umožňuje tak sdílení okolo 90% kódu napříč platformami a poskytuje dobrý výkon podobný nativním aplikacím. Podporuje funkci Hot Reaload, která umožňuje ihned zobrazovat změny provedené v XAML bez nutnosti opětovné kompilace [13].

Nevýhodou může být tvorba aplikací využívajících složitější grafiku. Další je malé množství balíčků pro manipulaci s hardwarem telefonu pro Xamarin.Forms. V těchto případech je nutné použít nativní kód.

Flutter

Flutter[16] je technologie od společnosti Google pro vývoj multiplatformních aplikací. Umožňuje vývoj pro Android, iOS, Web a desktop psaním jednoho sdíleného kódu. Aplikace jsou psány v jazyce Dart. Ve Flutteru se využívá práce s Widgety, což jsou základní grafické prvky, kterých obsahuje obrovské množství. Dále, stejně jako Xamarin, podporuje Hot Raload. Nevýhodou může být to, že se jedná o velmi mladý nástroj, který zatím není tolik rozšířený.

React Native

React Native[17] je open-source framework od společnosti Facebook pro psaní aplikací na Android a iOS. React je javascriptová knihovna k vytváření uživatelského rozhraní. React Native pak využívá upravené React komponenty, které překládá do nativních komponent pro cílovou platformu, kde je výsledný vzhled aplikace téměř identický. Také podporuje Hot Reloading. Jako nevýhoda může být nízký výkon grafických prvků a animací.

[16] https://flutter.dev
[17] https://reactnative.dev

Kapitola 3

Návrh řešení

Tato kapitola pojednává o problematice nezávislého návrhu řešení. Nejprve seznamuje čtenáře s problematikou a potřebami uživatele. V následující podkapitole se věnuje analýze problému, kde řeší, co by měla výsledná aplikace obsahovat, aby uspokojila potřeby uživatele. Dále je rozebrána architektura aplikace, kde jsou popsány její dekomponované moduly. Následující kapitoly se pak věnují návrhu řešení těchto modulů a návrhu uživatelského rozhraní.

3.1 Úvod do problematiky

Cílem je navrhnout a vytvořit nástroj sloužící k výuce a procvičovaní hry na bicí nástroje a rytmického čtení notace. Uživatelem je tedy bubeník, který potřebuje procvičovat své dovednosti. Zpravidla má sadu cvičení různých obtížností zapsaných v notovém zápisu. Cvičení mohou mít několik různých zaměření. Například cvičení na vířivý buben, která bývají více technická a jsou z větší části zaměřena na trénování rukou. Cvičení zaměřená na rytmické čtení notace, která mají za úkol procvičit orientaci v notovém zápisu. Dále můžou být cvičení na bicí soupravu, nohy, nebo perkuse.

Veškerá hudba (vyjdeme-li z definice, hudbou se rozumí organizovaný systém zvuků produkovaný člověkem), tedy skladby, rytmy, ale i etudy, se hrají v nějakém tempu. To je udávané v BPM a určuje, jak rychle se má skladba hrát. Při cvičení si hudebník volí tempo podle svých schopností, tak aby ho byl schopen zahrát. Po dostatečném procvičení, tempo zvyšuje. Zcela přesně udává tempo metronom.

Během učení je dobré mít nějaký dohled. Problémem domácího procvičování bez učitele je, že bubeník nemá zpětnou vazbu o tom, jestli cvičení hraje správně a toto zhodnocení závisí pouze na jeho úsudku, který nemusí být odpovídající realitě, což může vést k nepřesnému hraní a pěstování si špatných návyků. Tento problém samozřejmě existuje i pod dohledem učitele, který nemusí postřehnout (a pravděpodobně ani nepostřehne) všechny údery a jejich přesnost i když by se dalo předpokládat, že učitel bude odborník, který lépe rozpozná možné chyby a nepřesnosti. Uživatelem nejsou jen začátečníci ve výuce na bicí, ale bubeníci jakékoliv úrovně a dovedností, neboť neexistuje konečná hranice dovedností, které by bylo možno dosáhnout.

Další důležitou věcí, kterou ještě je potřeba o uživateli znát, je prostředí, kde trénuje. To závisí na možnostech a zaměření konkrétního bubeníka. Pokud bude potřebovat trénovat na celou soupravu, tak se mu nabízí možnost buď hrát na akustické nebo elektronické bicí nástroje. V prvním případě se jedná o možnost, která má spoustu omezení. Jednak je to

velice hlučný hudební nástroj, tudíž je potřeba hrát v prostorech k tomu přizpůsobených. Pokud bubeník bydlí v domě, kde nebude nikoho rušit, pak může na soupravu trénovat doma, v opačném případě bude nucen využít zkušebny nebo jiná místa, zaměřená pro hraní na hudební nástroje. U elektronických bicích odpadá problém vysoké hlučnosti a jsou mnohem méně prostorově náročné. Lze na ně tedy hrát i v bytě. Naopak u procvičování rukou a technik na vířivý buben je možnost využít tréninkového padu, který je díky svým rozměrům a nízké hlučnosti velice snadno přenosný a lze ho použít téměř kdekoli.

Potřeby uživatele

Není problémem si vzít s sebou noty a pad a cvičit na něj kdekoliv. Pro kvalitnější a přesnější hraní je nutné mít s sebou metronom. To také nepředstavuje problém, neboť existuje nepřeberné množství aplikací a lze tedy mít metronom v telefonu. Co naopak je problémem při cvičení nejen v terénu, ale i doma, je kontrola přesnosti hraní, která úplně chybí.

Uživatel při učení se nových technik nebo při procvičování chce vědět, zda cvičení podle notace zahrál správně a přesně a pokud ne, tak v jakých částech cvičení udělal chybu nebo byl nepřesný. Toto může pomoci řešit aplikace, která bude znát obsah tohoto cvičení, bude mu ho zobrazovat v notovém zápisu a dokáže uživateli podat zpětnou vazbu s výsledkem jeho cvičení.

Uživatel potřebuje všechny tyto věci a chce je mít přehledně na jedno místě:

- Chce trénovat kdekoliv
- Chce se učit
- Potřebuje notový zápis
- Potřebuje měnit tempo své hry
- Potřebuje používat metronom
- Chce mít informace o přesnosti zahraného cvičení

3.2 Analýza problému

Navrhovaná aplikace má sloužit jako nástroj k učení a procvičování. Bude zaměřená na převážně na čtení not. Nebude tedy řešit správně držení paliček, sezení u bicí soupravy nebo popisy jednotlivých bicích nástrojů a technik. Na co ale naopak bude zaměřená, jsou základní popisy not a převážně procvičení čtení těchto not.

Pokud má aplikace sloužit nejen k procvičování, ale i jako výuková aplikace, je nutné, aby obsahovala několik lekcí různých obtížností. Lekce musí obsahovat popis toho, co konkrétně se zde uživatel naučí čili učební text k dané lekci. Uživatel zde získá veškeré informace k tomu, aby zvládl cvičení k lekci přiložené a využívající prvky nových poznatků z učebního textu dané lekce. Toto cvičení bude zobrazeno v notách, podle kterých bude uživatel hrát.

Jak bylo řečeno v předchozí podkapitole, uživatel potřebuje cvičení přehrávat v nějakém tempu, a to si podle své potřeby zvyšovat nebo snižovat. Tím se navíc docílí volby obtížnosti i u jednotlivých cvičení. S tím souvisí metronom, který bude vydávat akustické i vizuální

signály. Sice existuje spousta aplikací metronomu, ale zde je důležité, aby byl obsažen přímo v této aplikaci, uživatel ho přímo viděl a mohl s ním manipulovat.

Nejdůležitější vlastností aplikace je bezpochyby podání informací uživateli o míře přesnosti jeho hraní. K tomuto účelu je potřeba, aby aplikace vytvořila audio záznam uživatelova hraní, v tomto záznamu vyhledala zahrané údery a ty pak porovnávala s hraným cvičením. Tím bude zjišťovat, jak uživatel hraje, jak se mu daří jednotlivé noty zahrát, jestli nějaké nevynechává nebo nehraje nějaké navíc. A pokud notu zahraje, tak jak moc přesně. K tomu, aby to bylo možné, je potřeba vyřešit několik problémů:

- Datová struktura, ve které bude cvičení uloženo
- Nahrání audiozáznamu
- Extrakce časů jednotlivých not a pomlk ve cvičení v závislosti na BPM
- Detekce úderů v nahrávce a časů jejich výskytu
- Porovnání časů not ve cvičení s detekovanými časy úderů

Aplikace bude muset umět nahrát audiozáznam, který analyzuje a vyhledá v něm údery a ty porovná se cvičením. Výsledkem této operce budou dvě metriky. Jedna reprezentuje přesnost zahrání jednotlivých not ve cvičení a druhá určuje poměr správně zahraných not vůči všem notám ve cvičení vyjádřený v procentech.

Dále by aplikace měla umožňovat ukládání výsledků do souboru, aby po zapnutí viděl skóre všech dokončených lekcí a mohl si zobrazit vyhodnocení svého posledního procvičování v dané lekci. Jistě se bude hodit i uložení posledního uživatelem zvoleného BPM, aby mohl pokračovat ve stejném tempu, ve kterém dané cvičení hrál naposled.

Po definování uživatelských potřeb a analyzování problému k řešení shrňme základní vlastnosti aplikace:

- Seznam lekcí různých obtížností
- Metronom s vizuální a akustickou signalizací
- Nahrávání audia
- Učební texty lekcí
- Cvičení lekce
- Porovnání nahrávky a se cvičením
- Zobrazení výsledků a jejich vizualizace v notách
- Uložení výsledků a nastavení do souboru

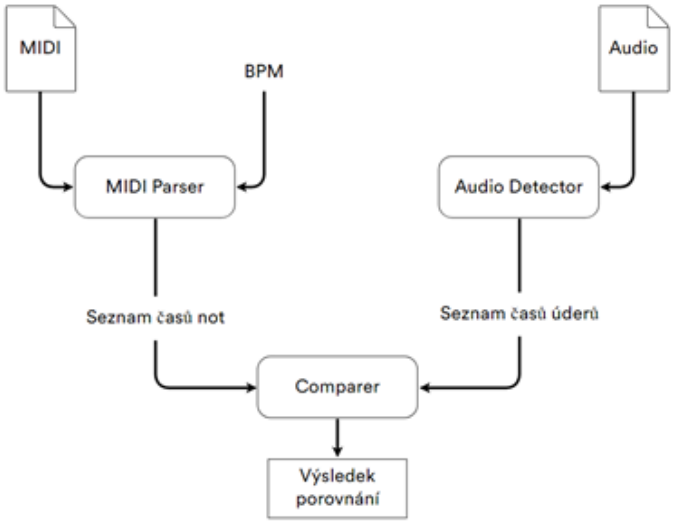
Uživatel tedy v aplikaci potřebuje vybrat lekci, zobrazit si její učební text, zobrazit cvičení k dané lekci, zahájit a ukončit cvičení, zobrazit si výsledky hraní a uložit je.

3.3 Architektura aplikace

Pro uložení not cvičení se dá použít několik způsobů. Lze například vytvořit vlastní datovou strukturu s informacemi o notě a časem jejího zahrání. Tento způsob by byl velmi jednoduchý, avšak není moc vhodný pro vytváření cvičení. Bude tedy výhodnější použít již nějaký existující formát. Jako vhodný se jeví použití formátu SMF (.mid) rozhraní MIDI. MIDI rozraní a formát souborů SMF je detailněji popsán v předchozí kapitole. Zde akorát shrnu výhody a důvod jeho použití. Hlavním důvodem je dlouhodobé používání MIDI standardu v hudbě. Lze v něm definovat jakoukoliv skladbu na jakékoliv nástroje. Existuje nepřeberné množství aplikací na vytváření či editaci SMF souborů. I když se ve cvičení naší aplikace budou vyskytovat pouze noty pro vířivý buben, práce s formátem MIDI je relativně jednoduchá, lze jednoduše měnit BPM a poskytuje spoustu možností rozšíření pro další vývoj.

Celou aplikaci lze rozdělit na několik vzájemně komunikujících modulů. Nejdůležitějším a nejsložitějším výpočetním modulem je modul pro zpracování nahraného audia a jeho porovnáním s přehrávaným cvičením. Tento modul dostane na vstupu SMF soubor s notací cvičení, uživatelem nastavené BPM a audio nahrávku hry uživatele. Výstupem je ve výsledek porovnání, tedy časový posun jednotlivých not vůči zadání a poměr počtu správně zahraných not vůči všem notám.

Funkce tohoto modulu se dále dělí na další části (obrázek3.1). Je potřeba zpracovávat MIDI soubory a vyhledávat v nich patřičné MIDI události a potřeba detekovat údery z nahraného audio souboru. Z toho důvodu tento hlavní modul obsahuje *MIDI Parser*, kde vstupem je již zmiňovaný SMF soubor a hodnota BPM a výstupem je seznam notových značek, kde každá nese informace o čase jejího zahrání a zda se jedná o notu nebo pomlku. Dále pak *Audio Detector*, který se stará o detekci úderů z audiosignálu, jehož vstupem je audio soubor a výstupem seznam časů, kdy jsou jednotlivé noty zahrané. Poslední částí je *Comparer*. Ten, jak název napovídá, slouží k porovnávání seznamů získaných z MIDI Parseru a Audio Detectoru. Výsledek porovnání je zároveň výstup celého modulu.



Obrázek 3.1: Schéma modulu pro detekci audia a porovnání

3.4 Detekce úderů a porovnávání se cvičením

Metoda slouží pro porovnání a hodnocení přesnosti zahraných úderů, které jsou nahrány v určitém tempu podle notového zápisu.

Problém porovnávání audiozáznamu se souborem MIDI lze rozdělit na několik částí. Je potřeba načíst časy událostí z MIDI souboru, v audio souboru rozpoznat jednotlivé údery a časy jejich výskytů a tyto časy porovnat s časy získané z MIDI. Na základě porovnání vytvořit hodnocení o úspěšnosti.

Čtení z MIDI

Ze souboru MIDI je potřeba načíst požadované události. Zajímat nás budou události zahrání noty (*NoteOn*) a ukončení zahrání noty (*NoteOff*) v případě, zda chceme brát v potaz i pomlky. Dále je potřeba z hlavičky přečíst informace o nastaveném tempu v BPM a počtu tiků na jednu čtvrťovou notu.

Každá *NoteOn* i *NoteOff* událost nese informaci o hrané notě a čase, kdy tato událost nastala. Ta je definována jako doba od poslední události v tikách. U toho musíme uchovávat počet tiků, o které jsme se již posunuli, jež se rovná jejich celkovému počtu od začátku souboru po aktuálně čtenou událost. Pokud tedy víme, kolik tiků trvá jedna čtvrťová nota a známe tempo v BPM, můžeme snadno dopočítat reálný čas každé události. Z toho lze poté i odvodit hodnoty jednotlivých not.

$$t = \frac{T_A}{\frac{BPM}{60 \times T_Q}} \tag{3.1}$$

Kde t – čas v sekundách, T_A – absolutní čas v tikách, T_Q – počet tiků za čtvrťovou notu.

Čtení úderů z audia

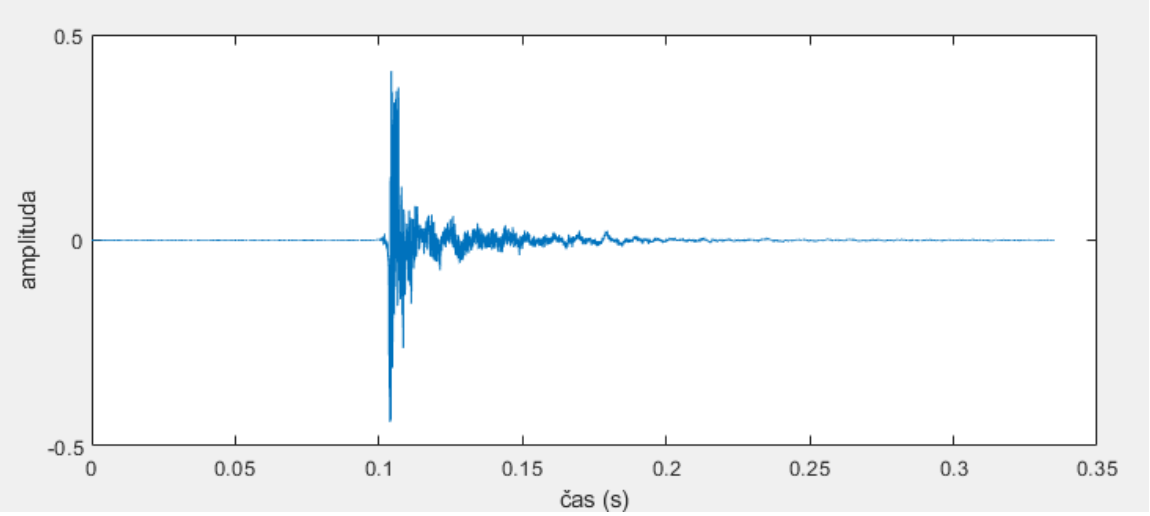
Hlavním problémem při poskytování zpětné vazby uživateli je detekce zahraných úderů. Úder je jedním ze způsobů, kterým vzniká zvuk. Ten se, mimo jiné, vyznačuje svou fyzikální intenzitou (hladina intenzity zvuku). Intenzitě pak odpovídá subjektivní veličina hlasitost.

K účelu detekce úderů je při přehrávání cvičení pořízena audio nahrávka. Předpokládá se, že nahrávka obsahuje záznam odehraného cvičení, tj. že obsahuje záznam úderů provedených paličkami na tréninkový pad nebo vířivý buben. Tyto údery je třeba v nahrávce rozpoznat a zjistit v jakých časech nahrávky se vyskytují.

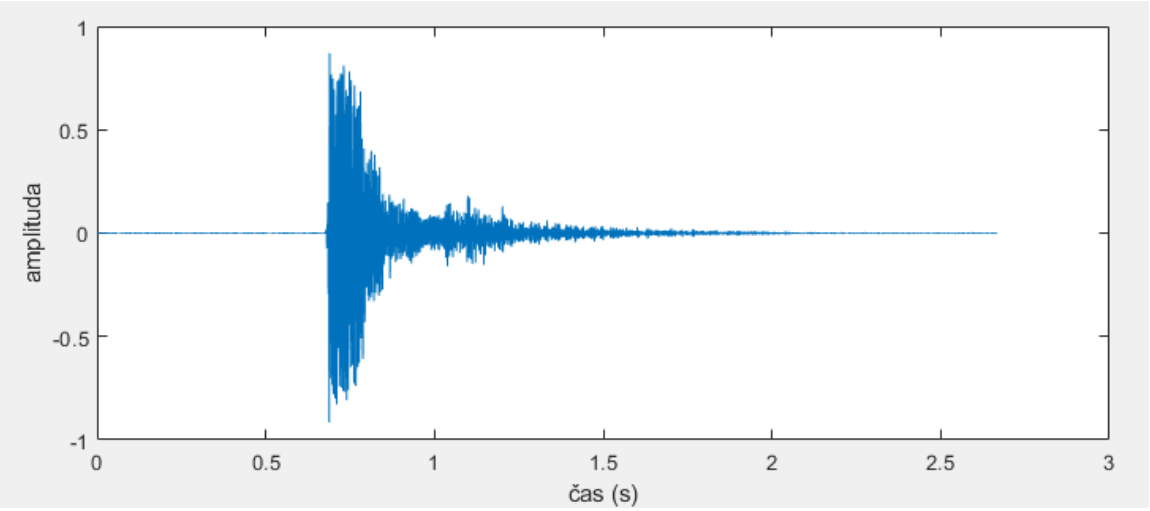
Detekce úderu

Zde vzniká náš problém, a to jakým způsobem detekovat údery v nahrávce. Ze zvukového hlediska je úder krátký zvuk, jehož hlasitost (energie) je výrazně vyšší než hlasitost okolních zvuků. Jinými slovy úder slyšíme, pokud mozek detekuje velké variace zvukové energie. V případě, že je hladina okolního hluku příliš vysoká, úder se v něm ztratí a nejsme schopni ho rozpoznat. Tyto vrcholy zvukové energie můžeme označit jako údery.

V audio souboru se budou detekovat variace energie výpočtem průměrné energie a porovnáním s okamžitou energií. Jako okamžitá energie se bude považovat energie obsažená v rámci o velikosti 25ms. Řekněme, že okamžitá energie bude vypočítána z 1024 vzorků. To při vzorkovací frekvenci 44100 Hz odpovídá přibližně 25ms. Pak se okamžitá energie vypočítá podle 3.2.



Obrázek 3.2: Průběh úderu do padu



Obrázek 3.3: Průběh úderu do vířivého bubnu

$$E = \sum_{k=0}^{1024} a\,[k]^2 \tag{3.2}$$

Výpočet průměrné energie by neměl být prováděn na celé nahrávce. Některé části nahrávky mohou obsahovat tiché části i údery zaznamenané velmi hustě za sebou. Okamžitá energie by měla být porovnávána s průměrnou energií v okolí jejího výskytu. Pokud bude nějaká pasáž obsahovat více nahuštěných úderů, nebude tato část ovlivňovat klidnější části. Pro vypočítání průměrné energie se využije vyrovnávací paměť, kam budeme ukládat historii okamžité energie, kde pouze spočítáme průměr posledních několika záznamů. Například, chceme-li vyrovnávací paměť 1s, vezme se pro výpočet posledních 43 záznamů, což odpovídá 44 032 vzorkům (44 100 je přesně jedna sekunda).

Pro nalezení úderu tedy musí platit:

$$E > C \times E \tag{3.3}$$

Kde C je vhodně zvolená konstanta citlivosti. V případě, že v nahrávce jsou obsaženy výrazné, dobře čitelné údery, C může být vyšší (třeba 1,4) – tím pádem méně citlivé. Pokud předpokládáme opak, C zvolíme nižší (např. 1,1), riskujeme tím ale, že mohou být za úder označeny i hlasitější pasáže.

Jelikož se hustota úderů může (a pravděpodobně bude) v nahrávce měnit, Bylo by vhodné používat pro každý usek jinou hodnotu konstanty C. Tu můžeme upravit na základě hodnot z vyrovnávací paměti. Spočítáme rozptyl jejich hodnot. Čím vyšší rozptyl bude, tím nižší citlivost zvolíme.

$$V = \frac{1}{N} \times \sum_{i=0}^{N} \left(E\left[i\right] - E\right)^2 \tag{3.4}$$

Analýza signálu probíhá po rámcích určité velikosti (např. 20ms). Při procházení audiozáznamu je třeba držet si informaci o tom, kolikátý rámec se analyzuje. Pokud je nalezený úder, na základě velikosti rámce se dopočítá, v jakém čase se nachází právě analyzovaný rámec. Začátek času tohoto rámce je pak považován za místo úderu.

Frekvenční analýza

Výše popsaný postup detekuje jen energetické vrcholy. Může nám tedy detekovat i parazitní zvuky, jako je třeba tlesknutí, ťuknutí paličkami o sebe atd. Dále, pokud nahrávka bude obsahovat velké množství okolního hluku, může se stát, že v něm úder splyne a nebude detekován.

Postup lze rozšířit na detekci velkých variací energií v konkrétních frekvenčních pásmech. To nám umožní detekovat nebo se zaměřit na údery v nich vyskytující, například chcemeli detekovat pouze snare. Postup zůstává stejný jako klasické detekci úderu, jen se místo výpočtu v časové doméně přesuneme do frekvenční domény. Tu získáme pomocí DFT a rozdělíme ji na několik frekvenčních pásem. Pro každé pásmo spočítáme okamžitou energii a porovnáme s průměrnou energií vyrovnávací paměti daného pásma. Pokud je energie jednoho nebo více pásem vyšší než jeho průměr, detekovali jsme úder. Další výhodou je, že si můžeme zvolit jen ta pásma, která nás zajímají. Nevýhodou pak vyšší časová a paměťová náročnost algoritmu.

Při návrhu této metody jsem vycházel z článku na webu *gamedev.net*, kde se popisuje postup vyhledávání beatů v hudbě [6].

Porovnávání

Detekované časy úderů se porovnávají vůči časům midi událostí. Cílem porovnání je zjistit, zda detekované časy úderů odpovídají časům událostí. Je třeba vzít v potaz, že uživatel nějaké noty nezahraje nebo naopak zahraje not více. Z toho důvodu není možné porovnávat časy jeden po druhém, ale pro každý čas události (noty) hledat, zda pro ni existuje ekvivalentní úder.

Je třeba brát v potaz, že pomlky nejsou v midi souboru zaznamenány jako událost a jako pomlka se musí brát úsek, kde žádná nota nehraje. To znamená, v čase události *NoteOff* není přítomna událost *NoteOn*. Délka doby do následující *NoteOn* události (nebo konce souboru) se rovná délce pomlky nebo kombinace více pomlk. Pomlky je potřeba také zaznamenat, aby se dalo posuzovat, zda v jejím čase nebyl zaznamenán úder.

Z jednotlivých časů not a pomlk z midi podle nastaveného BPM (viz kapitola Čtení z MIDI) se určí okolí noty – časový úsek, ve kterém se bude hledat existence zaznamenaného úderu. Pro určení okolí noty se na časové ose vyznačí středy mezi časy not a pomlk, které lze definovat jako průměr mezi dvěma sousedními notami. Každé okolí je tedy časový interval, od středu předchozí a aktuální noty po střed aktuální noty a noty následující.

Pro každé okolí noty se v seznamu detekovaných úderů hledá, zda existuje úder v čase vyhovující tomuto intervalu. U vyhovujícího úderu se dále posuzuje poloha (přesnost) vůči času noty, kde se spočítá posun. Jako výchozí tolerance byla zvolena pevná hodnota 200ms. Přesnost se počítá v procentech a to tak, že pokud je posun 0ms, přesnost se rovná 100% a na hraně tolerance, tj. pokud je posun 200ms, přesnost je 0%. Cokoliv mimo toleranci se počítá jako přesnost 0%. Pokud žádný úder nevyhovuje nebo jich vyhovuje více, je to bráno jako chyba a přesnost je rovněž 0%.

Okolí noty se stejně počítá i u pomlk. U porovnávání je zde však rozdíl a nebere se v potaz žádná tolerance. Tedy pokud v intervalu pomlky nebyl zaznamenán žádný úder, výsledek je 100%. Naopak pokud se zde nějaký úder vyskytuje, je to 0%.

Celkové hodnocení cvičení je počítáno jako poměr celkového počtu not ve cvičení a počtu správně zahraných not.

3.5 Testování metody detekce úderů

Při testování metody vznikají dva hlavní problémy. Jeden je metoda rozpoznávání úderů, kde nás bude hlavně zajímat, zda navržená metoda správně detekuje údery, a druhý problém je porovnávání samotné.

Testování detekce úderů

Pro testování detekce úderů je potřeba vytvořit sadu audio vzorků s nahrávkou úderů na různé typy vířivých bubnů a tréninkových padů. Sada by měla obsahovat nahrávky s různým počtem úderů. Bude se kontrolovat, zda detekovaný počet úderů odpovídá počtu úderů v nahrávce. Sada by mohla být dále doplněna o vzorky s přidaným šumem.

Testování porovnávací metody

Při testování porovnávací metody se použijí úseky cvičení v MIDI a k němu adekvátní audionahrávky, které by měli být relativně krátké, např. kolem 20 s. Nahrávky by měly být pořízeny několika způsoby s různými vlastnostmi:

- Bez chyb / s chybami
- Na snare / na pad
- Hrané bubeníkem / vygenerované z MIDI souboru

Výstupní hodnocení metody porovnávání pak probíhá s porovnáním výsledného hodnocení s hodnocením očekávaným.

3.6 GUI aplikace

Po definování všech funkcí, kterými má aplikace disponovat, je možné navrhnout, jak bude aplikace vypadat, jaké bude mít kontrolní prvky a jak budou rozmístěné. Je potřeba si říct,

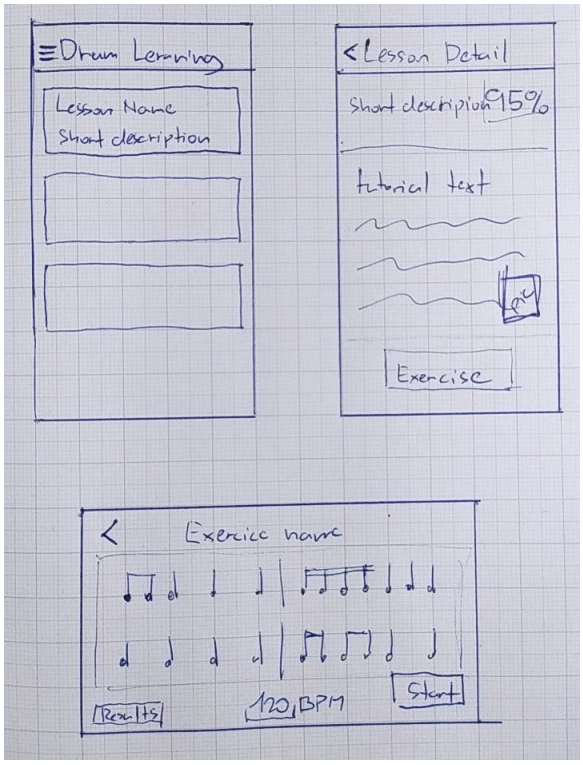
jak uživatel bude s aplikací pracovat, s jakými daty bude muset manipulovat a co přesně potřebuje vidět.

Aplikace obsahuje přímo vestavěná data, která jsou neměnná. Těmito daty jsou u každé lekce její popis, učební text, soubor SMF a notace cvičení. Jediná data od uživatele jsou audio nahrávka cvičení a nastavení tempa. Uživatel pak potřebuje tato data zpracovat, zobrazit si výsledek porovnání a mít možnost jej uložit.

Uživatel musí mít možnost zobrazit seznam lekcí, ze kterých si může vybírat. Každá lekce musí zobrazovat svůj vlastní název, krátký popis a učební text. Dále musí obsahovat cvičení, které lze spustit a zastavit, zobrazovat jeho notový zápis, nastavit si tempo a zobrazovat výsledky.

Skicování a mockupy

Pro rychlý náhled a představu, jak by mohla aplikace vypadat lze využít tzv. ruční skicování. Používá se pro rychlou vizualizaci prvního návrhu samotné aplikace. Při vytváření skic, lze vidět nedostatky v návrhu a může nám pomoci k lepším nápadům na vylepšení aplikace a jejího rozvržení. U návrhu aplikace byla použita tato metoda, ze které vyplynulo, že je vhodné GUI aplikace rozdělit na celkem tři základná stránky, na které lze rozmístit všechny důležité prvky. Na obrázku 3.4 jsou vidět náčtrty výsledného grafického rozhraní a obrázky 3.5 a 3.6 ukazují mockup, vytvořený na základě této skici.



Obrázek 3.4: Skica návrhu rozložení jednotlivých stránek aplikace

První a zároveň hlavní stránkou bude seznam všech lekcí. Každá položka seznamu bude zobrazovat název lekce a její krátký popis.

Druhou stránkou bude detail lekce, na který se se uživatel dostane rozkliknutím položky seznamu. Tato stránka bude obsahovat veškeré informace o lekci. Rozhodně by se hodilo zde

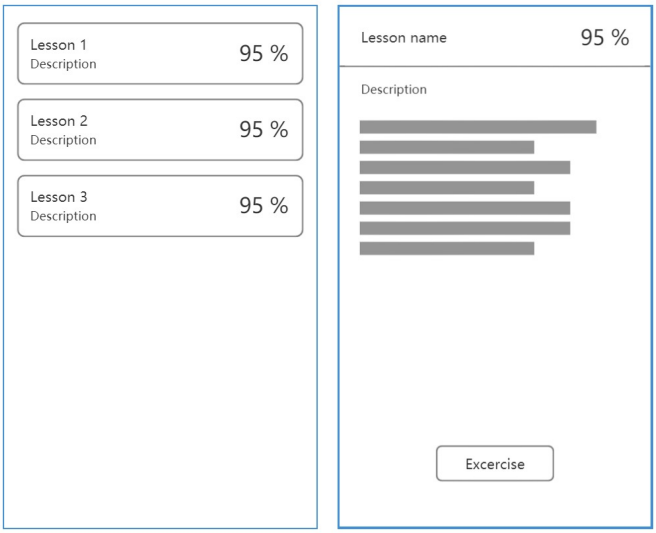
mít informaci o nejlepším dosaženém skóre. Největší část obrazovky bude obsahovat učební text. Z něho se uživatel dozví veškeré informace a nové poznatky, které bude procvičovat ve cvičení. Posledním prvkem na této stránce bude tlačítko k přechodu k procvičování.

Obrazovka se cvičením bude pro uživatele určitě ta nejdůležitější. Zde se nachází jádro celé aplikace. Nejprve shrnu, jaké prvky zde budou muset být zobrazeny.:

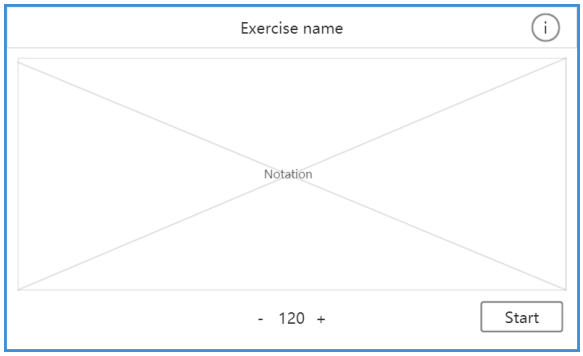
- Zobrazení not
- Pole pro zadání BPM
- Tlačítko start a stop
- Vizuální metronom
- Vizualizace výsledků v notách
- Tlačítko pro zobrazení posledních výsledků

Největší plochu bude zabírat zobrazení not, je potřeba, aby byly čitelné i na menším displeji. Do not se bude zobrazovat i vizuální výsledek porovnání. To bude řešeno barevným označováním jednotlivých not tak, že správně zahrané noty a pomlky budou označeny zelenou barvou a ty špatně zahrané budou označeny červeně.

Ostatní ovládací prvky budou rozmístěny u horního a dolního okraje obrazovky. Tyto okraje by měli zabírat co nejméně místa, ale s ohledem na přehlednost a nutnou velikost prvků na nich umístěných. Na dolním okraji se bude vyskytovat tlačítko pro zahájení cvičení prvek pro nastavení BPM. Na horním bude název cvičení a tlačítko pro zobrazení výsledků.



Obrázek 3.5: Mockup GUI; vlevo – seznam lekcí, vpravo – detail lekce



Obrázek 3.6: Mockup GUI – Stránka cvičení

3.7 Datové struktury

U každé aplikace je důležité zajistit uchování dat i po jejím vypnutí. Je potřeba zvolit, jaká data budou muset být uložena. V tomto návrhu je datová struktura velice jednoduchá. Jedná se o data s informacemi o lekcích a cvičeních. Těmito daty jsou:

- ID
- Název lekce
- Popis
- Učební text
- MIDI soubor
- Soubor s notací

Každá lekce musí mít unikátní identifikátor k jejich vzájemnému odlišení, název, popis a učební text. Data zobrazující notaci a MIDI soubor budou uloženy zvlášť ke každé lekci. Data lekcí a cvičení je potřeba doplnit uživatelskými daty, která půjde modifikovat.

- Nastavené BPM
- Nejlepší dosažení hodnocení
- Podrobné hodnocení jednotlivých zahraných not

Je potřeba zachovat informace o nejvyšším dosaženém hodnocení. Protože uživatel bude pravděpodobně cvičení hrát pokaždé v tempu blízkému tomu z předchozího hraní, je potřeba ukládat i hodnotu naposledy nastaveného BPM. K tomu, aby si uživatel mohl zobrazit poslední výsledky, je potřeba ukládat i je.

Informace o notách cvičení jsou sice reprezentovány souborem MIDI, ze kterého se čtou časové značky obsažených událostí, ale pro účely porovnání je potřeba více informací. Je potřeba určit interval, ve kterém se má detekovaný úder vyskytovat. Data s notami budou reprezentovány strukturou obsahující tyto informace:

- Začátek a konec intervalu

- Čas noty
- Typ (nota / pomlka)
- Číslo noty
- Délka trvaní noty

Kapitola 4

Aplikace Drum Learning

V této kapitole je popsán způsob implementace výsledné aplikace. Na začátku je popsána volba použité technologie. Následuje podkapitola popisující architekturu celé aplikace a popis jejích částí. Dále je část věnující se grafickému uživatelskému rozhraní. Věnuje se popisu výsledného vzhledu a vizuálních prvků aplikace a technologiím při tom použitých. Nakonec je jedna kapitula věnována implementaci knihovny pro práci s audiem.

4.1 Volba cílové platformy a použitého frameworku

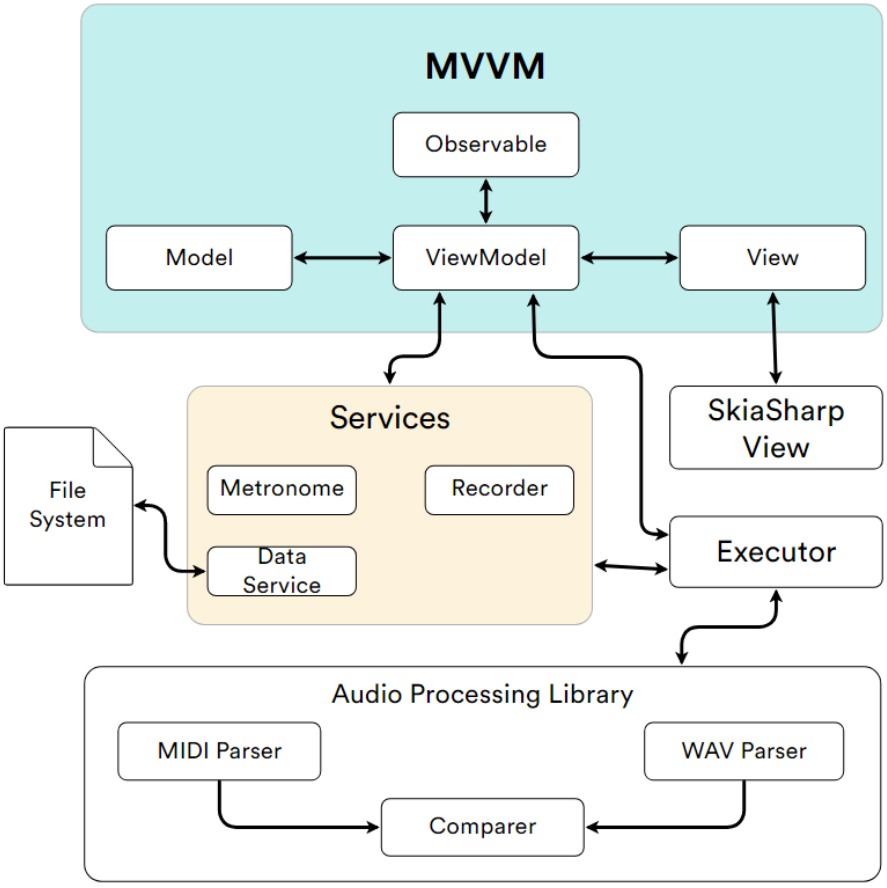
V předchozí kapitole bylo popsáno, kdo je uživatel a v jakém prostředí potřebuje aplikaci používat. Víme, že bude potřeba umožnit mu mít aplikaci k dispozici na jakémkoliv místě. Z toho důvodu je potřeba implementovat aplikaci pro mobilní zařízení. Dále víme, že v mobilním světe dominují dvě platformy, a to Android a iOS. Aby tato aplikace byla přenosná a mohlo ji používat co nejvíce uživatelů, byl zvolen multiplatformní vývoj ve frameworku Xamarin, konkrétně s využitím technologie Xamarin.Forms. To umožní sdílet jediný kód logiky aplikace napsaný v C# a použitím .NET i uživatelské rozhraní aplikace ve značkovacím jazyce XAML mezi obě zvolené platformy.

4.2 Architektura aplikace

Při implementaci byl kladen důraz na přehlednost a znovupoužitelnost kódu. Tento přístup je důležitý pro budoucí práci a rozšiřování funkcionality aplikace. Aplikace je tvořena z několika vzájemně komunikujících bloků. Základ je postaven na architektonickém vzoru MVVM. Logika aplikace obsahuje několik služeb, mezi které patří správa dat a obsluha práce s audiem. Architektura aplikace je znázorněna na obrázku 4.1, kde jsou znázorněny všechny podstatné části aplikace a jejich vzálenmé vazby.

Model-View-ViewModel

Použití technologie Xamarin.Forms přímo vybízí k použití architektonického vzoru MVVM. Model-View-ViewModel byl navržen pro WPF aplikace a používá se i v Xamarin. Tyto technologie jsou přímo vytvořené tak, aby v nich bylo použití vzoru MVVM jednoduché a uživatelsky přívětivé. Jedná se o rozšíření vzoru MVC (Model, View, Controller) a slouží k oddělení dat, logiky a stavu aplikace a uživatelského rozhraní[4].



Obrázek 4.1: Architektura aplikace

- **Model** – pouze popisuje data, se kterými aplikace pracuje. Nesmí o stavu ovládacích prvků nic vědět.
- **View** – reprezentuje uživatelské rozhraní v jazyce XAML. Popisuje veškeré grafické prvky, které uživatel vidí. Muže to být celá stránka nebo jen jednotlivé prvky.
- **ViewModel** – je nejdůležitější vrstvou této architektury. Spojuje *Model* a *View* a drží si stav aplikace. Poskytuje všechna data, které potřebuje uživatelské rozhrání, zajišťuje transformaci dat mezi modelem a pohledem a zpracovává veškerou logiku. Důležité je, že uchovává data v takových strukturách, které vyvolávají události, kdykoliv se změní jejich hodnota. To díky bindingu (popsaný dále) umožňuje uživatelskému rozhraní na to reagovat a ihned zobrazit nová data.

V pochopení architektury MVVM je důležitý návrhový vzor Observer (pozorovatel). Ten umožňuje pozorovatelům objektu aktualizovat svůj stav, pokud se změní pozorovaný objekt. To v architektuře MVVM usnadňuje tok dat mezi jednotlivými vrstvami. Mezi *Observable* objekty patří zmíněný *VievModel*, ale jako *Observable* může být označený jakýkoliv jiný objekt, který je potřeba sledovat a reagovat na jeho změny.

V této aplikaci kromě *ViewModelů* figuruje právě ještě třída `ExerciseObservable`. Ta si drží informace o lekci a cvičení, jejích popisu, hodnocení a cestách k potřebným souborům.

Obvykle by tato data byla součástí *VievModelu*. Tato data ale využívá více pohledů, proto je jednodušší a přehlednější použít observable třídu, na kterou se odkazují ViewModely, které tato data využívají. Zde se jedná o `ExerciseDetailViewModel` a `ExerciseRunViewModel`. Ty kromě odkazu na `ExerciseObservable` obsahují už jen *Commandy* a vlastnosti s daty potřebné pro konkrétní *View*.

Zmíněný binding je další součástí vzoru MVVM. Slouží k provázání *ViewModelu* a *View*. Je možné spravovat data odděleně od jejich zobrazování. Každému grafickému prvku lze nastavit tzv. *Binding Context*, což je odkaz na *ViewModel*, ze kterého bude probíhat tok dat. Binding umožňuje tento tok dat v obou směrech. To nejen umožňuje okamžitě reagovat na změnu dat ve *ViewModelu* a ty vykreslit, ale i předávat data zpět *ViewModelu*, když například uživatel zadá nějakou hodnotu. Programátor může volit z několika režimů, ve kterém má binding pracovat [12]:

- **One-way** – Obnoví ovládací (View) prvek ve chvíli, kdy je změněna zdrojová vlastnost (ViewModel).
- **One-way to source** – Obnoví zdrojovou vlastnost ve chvíli, kdy je změněn ovládací prvek.
- **Two-way** – Kombinace předchozích dvou režimů. Ovládací prvek reaguje na změnu zdrojové vlastnosti a obráceně.
- **One time** – Obnoví ovládací prvek pouze ve chvíli, kdy je přiřazena zdrojová vlastnost. Na další změny vlastnosti už nereaguje.

Audio Recorder

Důležitým prvkem v této aplikaci je zaznamenání uživatelovy hry pomocí audiozáznamu. K tomuto účelu slouží tato služba, reprezentovaná třídou `Recorder`.

Jak bylo řečeno na začátku této kapitoly, multiplatformní aplikace v Xamarinu umožňují sdílet velké množství kódu mezi několika platformami. Použití periférií jako je mikrofon, kamera, úložiště atd. vyžaduje ale jednak povolení k jejich přístupu a dost často, kvůli rozdílným API u Androidu a iOS, také použití knihoven určených pro konkrétní platformu. V těchto případech nelze využít sdílený kód.

Při používání periférií je tedy nutné definovat požadované oprávnění. A to pro každou platformu zvlášť. Výpis 4.1 obsahuje ukázku definice oprávnění pro přístup k mikrofonu a úložišti na platformě Android. Dále je důležité zkontrolovat, zda aplikace má oprávnění k periférii, ke které chce přistupovat. Pokud by se ji pokusila použít bez oprávnění, vedlo by to k pádu aplikace. K ověření přidělených oprávnění slouží třída `Permissions` v knihovně *Xamarin.Essentials*. Tato třída umožňuje kontrolu oprávněni, a vyvolat zobrazení požadavku k jejich přidělení. Navíc je multiplatformní a lze ji tedy použít ve sdíleném kódu.

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android">
   <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
   <uses-permission android:name="android.permission.MODIFY_AUDIO_SETTINGS" />
   <uses-permission android:name="android.permission.RECORD_AUDIO" />
   <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
   <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
</manifest>
```

Výpis 4.1: Definice uprávnění přístupu k hardwaru na platformě Andriod

Třída Recorder využívá knihovnu *Plugin.AudioRecorder*, která je napsaná pro *Xamarin.Forms*, z toho důvodu stačí pouze jedna implementace nahrávání audia pro obě platformy.

Data Service

Služba sloužící k manipulaci s daty. Je reprezentovaná třídou `DataService`, která implementuje generické rozhraní `IDataService<T>` Jako generický typ bere třídu reprezentující data. Veškerá data související s lekcí jsou definována v modelu `ExerciseItem`. Z podstaty této aplikace, protože se jedná o výukovou aplikaci, však existují dva druhy dat:

- **Data lekcí** – jsou data, kam patří název lekce, její popis, cesta k souborům cvičení (.mid, .png) a její učební text. Tato data jsou neměnná a uživatel nemá možnost je jakkoli měnit ani přidávat.
- **Uživatelská data** – jedná se o data, která se mění s interakcí uživatele. Sem patří hodnocení, nastavené BPB, výsledek porovnání atd. Tyto data je potřeba modifikovat a ukládat.

Data lekcí drží třída `ExerciseDataStore`. Ta obsahuje kolekci všech lekcí. Pro účely této aplikace byly jako učební texty, náhledy cvičení i popisy jednotlivých lekcí použity články[1] od Normana Weinberga[2], profesora hudby a ředitele Studií bicích nástrojů na Arizonské univerzitě.

Metronome

Jak název napovídá, tato služba slouží k ovládání metronomu. V této aplikaci je potřeba, aby metronom umožňoval jak akustickou, tak vizuální signalizaci v pravidelném, předem nastaveném intervalu. Služba je tvořena třídou `Metronome` a jedná se v podstatě o rozšíření třídy `Timer`, kterou využívá. `Timer` umožňuje vyvolávat události v předem nastaveném intervalu.

Události umožňují třídě nebo objektu upozornit jiné třídy nebo objekty, když dojde k vyvolání události [9]. V .NET jsou založené na modelu delegáta, který dodržuje vzor návrhu Observable (pozorovatele). Pro práci s událostmi v C#, jsou potřeba dva pojmy:

- **Delegáti** – jedná se o typ, který obsahuje odkaz na metodu.
- **Události** – jsou to kolekce delegátů. Pokud dojde k vyvolání události, spustí se všechny metody odkazované delegáty.

Třída Metronome po zavolání metody `Start` reaguje na události `Timeru` přehráním audia a obsahuje metodu `AddElapsedMethod`, která umožňuje přidání dalších metod reagujících na událost. Toto je důležité, aby se třída mohla použít na spuštění obslužné rutiny pro vykreslení signalizačního prvku, aniž by bylo nutné porušit vzor MVVM. Takhle zůstane logika metronomu zapouzdřena v jedné třídě.

[1] https://vicfirth.zildjian.com/education/norm-weinberg-webrhythms.html
[2] https://www.normanweinberg.com

Executor

Jedná se o jednoduchou třídu obsahující pouze metodu `Run`. Slouží k obsluze knihovny *AudioProccesing*. Má přístup k instanci `ExerciseObservable` a `Recorder`, ze kterých si vezme informace potřebné pro výpočet výsledku uživatelova hraní. K tomu potřebuje zadanou hodnotu BPM, cestu k souboru MIDI cvičení a audiozáznam uživatelova hraní. Výsledek porovnání a hodnocení poté předá opět třídě `ExerciseObservable`, a jelikož je tato třída *Observalbe*, dojde okamžitě k aktualizování pohledů. Knihovna `AudioProcessing` je popsána v samostatné kapitole.

Kontejner – Inversion of Control

V této aplikaci jsou třídy, u nichž je vhodné nebo vyloženě požadováno, aby se vyskytovali pouze v jediné instanci, sdílené skrze celou aplikaci. V tomto ohledu se nabízí použití návrhového vzoru *Singleton* (jedináček). Vzor je tvořen třídou, která zajišťuje, aby její instance existovala v celé aplikaci jen jednou a zajišťuje k ní globální přístup. Požívá se v případě, když potřebujeme sdílet jednu instanci mezi několika objekty, aniž bychom si ji museli předávat v konstruktoru. Použití tohoto návrhového vzoru je však poměrně kontroverzní, a to kvůli nutnosti použití globální statické proměnné a složitému předávání závislostí.

V aplikaci byla použita knihovna Autofac[3]. Jedná se o *Inversion of Control* kontejner, který slouží k automatickému vkládání závislostí. Spravuje vytváření instancí objektů a jejich životnost. Řeší problém s předáváním závislostí. Všechny výše zmíněné služby se hodí mít jen v jedné instanci, proto jsou zaregistrovány jako *Singleton* právě v tomto kontejneru. Pokaždé, když je instance potřeba, je možné ji z kontejneru vyzvednout.

V aplikaci je kontejner obsažen ve třídě `AutofacIoc`, která je kvůli přehlednosti a lepší budoucí použitelnosti v samostatném projektu. Každý projekt, který chce používat tento kontejner, musí implementovat třídu dědící z `Autofac.Module` s metodou `Load`, ve které zaregistruje požadované typy. Inicializuje se při startu aplikace, kde se registrují tyto moduly.

4.3 Grafické rozhraní

V předchozí kapitole byl popsán návrh uživatelského rozhraní a ukázány mockupy jednotlivých stránek. V architektonickém vzoru MVVM je uživatelské rozhraní definováno pomocí pohledů s použitím značkovacího jazyka XAML. Ten byl použit pro sestavení jednotlivých stránek.

XAML

XAML (eXtensible Application Markup Language) je jazyk založený na jazyce XML vytvořený jako alternativa ke psaní kódu pro vytváření instancí objektů a uspořádání těchto objektů v hierarchiích nadřazených a podřízených objektů. Umožňuje uživateli definovat uživatelské rozhraní pomocí značek místo kódu. *Xamarin.Forms* definuje spoustu uživatelských prvků pro použití v XAML[11]. Na ukázce 4.2 je zobrazeno hierarchické seskládání prvků a definici prvku *StackLayout*, který slouží pro uspořádání do něj vložených prvků a prvek *Label* pro vykreslení textu.

[3] https://autofac.org

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="DrumLearning.Views.AboutPage"
    <StackLayout>
        <Label Text="Text"/>
    </StackLayout>
</ContentPage>
```

Výpis 4.2: Ukázka jazyka XAML

Každá definice uživatelského rozhraní v XAML je reprezentována dvěma soubory. Souborem .xaml, a přidruženým souborem .cs s kódem v jazyce C# [14]. Psaní logiky do tohoto souboru však porušuje principy vzoru MVVM, kde k tomuto účelu slouží *ViewModel*.

Vzhled aplikace

Jelikož aplikace slouží pro výuku, cvičení v ní obsažená jsou primárním prvkem k zobrazení. Uživatelské rozhraní je jednoduché a minimalistické. Barva rozhraní o tohoto typu aplikace nehraje příliš velkou roli. Důležité je, aby byla konzistentní a jednotlivé prvky a texty dobře čitelné. Nakonec byla zvolena modrá, která nepůsobí nijak křiklavě a rušivě.

Barva je definována v *Resource Dictionary*. To slouží jako slovník a používá se ke sdílení konstant. Zde se definují styly různých grafických prvků, které lze využívat v jazyce XAML napříč celou aplikací. To umožňuje jednu definici stylu a zachování konzistentního vzhledu aplikace. V této aplikaci jsou použity pro definování výchozích barev a výchozího stylu tlačítka.

Hlavní obrazovka (obrázek 4.2) obsahuje pouze seznam všech cvičení a vysouvací menu. Jednotlivé prvky seznamu bylo potřeba navrhnout tak, aby se do nich vešel základní popisek lekce, číslo lekce a maximální dosažené hodnocení. Jako barva textu hodnocení byla zvolena zlatá.

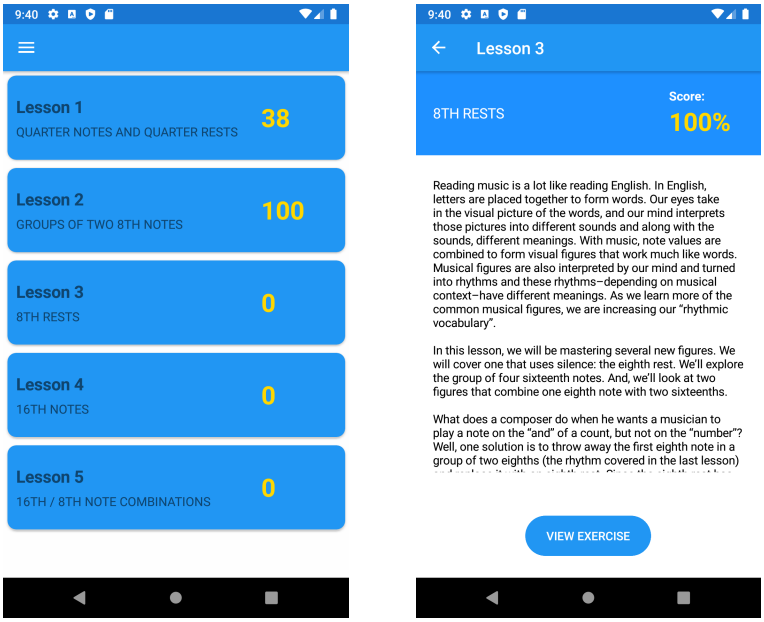
Obrazovka detailu (obrázek 4.2) lekce obsahuje učební text dané lekce. V horní části se nachází pruh s názvem lekce a hodnocením v primárních barvách a ve spodní je tlačítko k přechodu ke cvičení. Uprostřed stánky se nachází učební text s obrázkem přibližujícím danou problematiku.

Obrazovka cvičení

Nejdůležitější obrazovkou je pohled cvičení. Na této obrazovce jsou zobrazeny noty, spouští se nahrávání, nastavuje BPM a zobrazují výsledky porovnání. Na této obrazovce byl kvůli prostoru a přehlednosti skryt barevný navigační panel.

U zobrazování not je třeba brát v potaz velikost obrazovky. Jelikož se jedná o aplikaci na mobilní zařízení, jsou jen omezené možnosti. Pokud je telefon otočený na šířku, není problém zobrazovat dva řádky notace, kde se vejdou cca dva takty. Pří zobrazení na výšku by to vyžadovalo zobrazovat každý takt na jeden řádek. To není pro čtení moc příjemné. Proto celá tato stránka bude mít vynucené zobrazení na šířku, protože zobrazení na výšku se nezdá být vhodné. Do not se zároveň bude se vykreslovat výsledek porovnání. To je řešeno vytvořením vlastního prvku s využitím knihovny SkiaSharp. Způsob práce a implementace vykreslování výsledků je popsána dále v samostatné podkapitole.

Obrázek 4.3 ukazuje vzhled stránky se cvičením. V dolní části uprostřed je prvek pro zadání tempa. Uživatel může zadávat tempo přímo pomocí klávesnice. Pro lepší uživatelskou



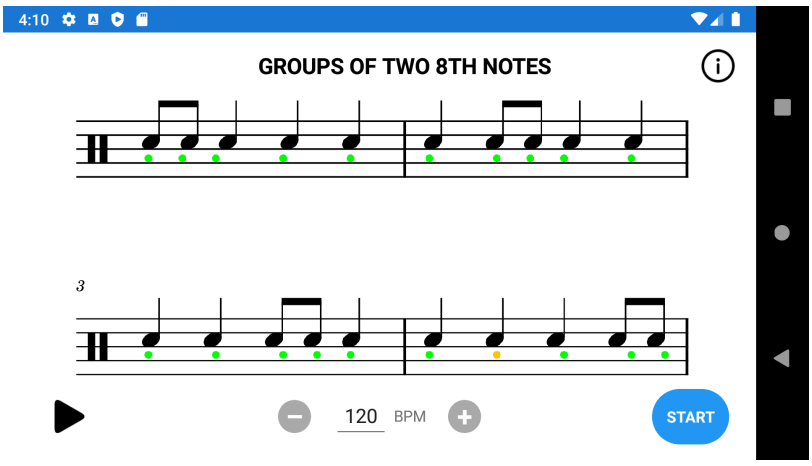
Obrázek 4.2: Vlevo – seznam lekcí, vpravo – detail lekce

přívětivost jsou k dispozici tlačítka plus a mínus pro rychlejší úpravu nastavené hodnoty o menší krok.

Tlačítko *Start* a *Stop* slouží k zapnutí a vypnutí metronomu a spuštění / stopnutí nahrávání uživatelova vstupu. Metronom je vizualizován pomocí tmavě červeného puntíku, blikajícího na obrazovce v nastaveném tempu, umístěného v levém horním rohu.

V levém dolním rohu se nachází tlačítko *Play*. To slouží k přehrání ukázky daného cvičení v rychlosti podle nastaveného BPM. Uživatel si tak může poslechnout, jak rytmus zní.

V pravém horním rohu se nachází tlačítko pro zobrazení posledních výsledků. Po kliknutí na něj zobrazí detailní výsledky porovnávání jednotlivých not, získané při posledním cvičení. Zobrazení je realizování pomocí dialogového okna.



Obrázek 4.3: Obrazovka cvičení s výsledkem

Vykreslování ve SkiaSharp

SkiaSharp[4] je multiplatformní 2D grafické API určené pro platformu .NET. Je založené na knihovně Google's Skia Graphics. Poskytuje komplexní 2D API, které lze použít na vykreslování vektorové grafiky, rastrových obrázků a textu. Knihovna umožňuje tvorbu *SVCanvasView* objektů. Jedná se o *View*, ze kterých lze odvodit vlastní třídu implementující vykreslování grafických prvků [10].

Problémem použití *SVCanvasView* objektů je složité dodržení vzoru MVVM. Z toho důvodu je vykreslování výsledku rozděleno do dvou tříd. Vlastní ovládací prvek reprezentovaný třídou `SKNoteView`, založený na *SKCanvasView*, a třída *NoteRender*, obsahující logiku pro vykreslování.

Třída `SKNoteView` slouží pro definici grafického prvku. Je možné ji přímo vkládat do XAML kódu a pracovat s ní běžným způsobem jako s ostatními prvky. Tento prvek je však potřeba překreslovat v reakci na získání výsledků. K tomuto účelu je do třídy přidána vlastnost *Renderer*, která je nastavena jako *BindableProperty*. Dále pak prvek obsahuje metodu *RendererRefreshRequest*, která se volá pokaždé, když se hodnota vlastnosti `Renderer` změní a spustí překreslování celého prvku. Tento způsob umožnuje použití jednoho grafického prvku, na který lze *bindovat* specifický *Renderer*.

Na ukázce 4.3 lze vidět použití prvku *SKNoteView* s přidanou vlastností `Renderer`, na kterou je nabindovaný objekt `NoteRenderer` obsahující logiku pro vykreslování prvku.

```
<ctrl:SKNoteView Grid.Row="1" x:Name="NoteView"
                 Renderer="{Binding NoteRenderer}"
                 VerticalOptions="FillAndExpand"
                 HorizontalOptions="FillAndExpand"
                 Margin="10, 0"/>
```

Výpis 4.3: Použití prvku založeného na SkiaSharp

Jako *Renderer* pro vykreslování notového základu a výsledku porovnání se používá právě třída `NoteRenderer`. Obsahuje vlastnosti `ImageFileName` – název souboru s obrázkem not, `CompareResult` s výsledky (popsané v následující kapitole) a `CanDrawResult` pro určení, kde se má výsledek vykreslit. Pro zasílání zpráv o změně hodnot, třída obsahuje událost `RefreshRequest`, která je vyvolaná při změně jakékoliv vlastnosti. K samotnému vykreslení slouží metoda `PaintSurface`. V ní se nejprve vykreslí obrázek s notami pomocí metody `DrawBitmap` a pokud jsou k dispozici výsledky porovnání, volá se metoda `DrawResult` vykreslující výsledek na základě vlastnosti `CompareResult`. Výsledky se vykreslují jako barevné puntíky označující jednotlivé noty.

`NoteRenderer` je vlastností *ViewModelu* a je nabindovaný na grafický prvek `SKNoteView`. Výsledek porovnání se po jeho vypočtení pouze předá této vlastnosti, a tak dojde k jejímu automatickému překreslení. Tímto způsobem jsou všechny třídy přehledně odděleny a je dodržen vzor MVVM [3].

4.4 Knihovna pro práci s audiem

Tato aplikace potřebuje pro svoji funkci zpracovávat audiozáznam, detekovat v něm zahrané údery a zjišťovat, zda se vyskytují ve správných časech. Pro každé cvičení je nutné, aby

[4] https://github.com/mono/SkiaSharp

obsahovalo právě tyto časové údaje. Proto je, mimo jiné, reprezentováno i souborem MIDI, který byl podrobně popsán v předchozí kapitole. Knihovna pro korektní vypočtení výsledku potřebuje onen zmíněný MIDI soubor, hodnotu BPM a audiozáznam ve formátu WAV.

Pro zpracování audia, práci s MIDI a porovnávání slouží knihovna `AudioProcessing`. V aplikaci je vytvořena jako samostatný projekt využívající framework .NET Standard 2.0. Lze ji tedy používat zcela samostatně. Toto se hodí nejen pro přehledné oddělení funkcionality, ale i pro snadné testování a případné další použití v jiných projektech. Knihovna obsahuje tři hlavní části:

- **MIDI Parser** – slouží k extrakci not z MIDI souborů
- **WAV Parser** – slouží ke zpravování audia a detekci úderů
- **Comparer** – porovnává seznamy časových značek získaných z MIDI a WAV parserů

Algoritmy popisující metodu detekce úderů a porovnávání se souborem MIDI byly popsány v kapitole 3.4. Dále se zde budeme věnovat způsoby jejich implementace.

Fasáda

Komunikace s knihovnou probíhá pouze přes její třídu `AudioProcessingFacade`, která je kromě modelů definující výsledky jedinou veřejnou třídou této knihovny. Je implementována podle návrhového vzoru Fasáda (Facade). Tento návrhový vzor se používá k vytvoření jednotného rozhraní pro celou logickou skupinu tříd, které se tak druží do subsystému. Fasáda se implementuje jako třída, která fasádu tvoří. Ta je napojena na ostatní třídy, se kterými pracuje. Z venku je viditelná pouze tato fasáda a cela složitá struktura tříd je v pozadí.

WAV Parser a detekce úderů

K účelu detekce úderů je při přehrávání cvičení pořízena audio nahrávka ve formátu WAV. Tento formát je zvolený z důvodu, že obsahuje nekomprimovaný jedno- nebo vícekanálový zvuk kódovaný pomocí pulzně kódové modulace. Data obsahují jednotlivé vzorky signálu vhodné k přímému zpracování.

Jak je z názvu tohoto modulu patrné, *WAV Parser* je modul, který pracuje s audio soubory WAV, které analyzuje, a vyhledává v nich zahrané údery.

WAV soubor má v hlavičce informace o vzorkovací frekvenci, počtu kanálů, velikosti vzorku atd. K K práci s WAV souborem se používá knihovna NAudio. Ta poskytuje třídu `WaveFileReader`, který načte WAV soubor a z hlavičky přečte všechny potřebné informace a umožňuje přístup k datům po bajtech. Tento způsob není moc vhodný, protože v případě šestnáctibitového kódování je jeden vzorek uložený na dvou bajtech, které by bylo nutné správně číst a interpretovat. Pro lepší práci s audio signálem je v knihovně *NAudio* rozhraní `ISampleProvider` do kterého lze `WaveFileReader` převést. Rozhraní obsahuje metodu `Read`, která umožňuje načítat jednotlivé vzorky jako hodnotu amplitudy.

Třída `WavParser` má dva konstruktory, jeden bere v parametru cestu k souboru a druhý datový proud (Stream) a obsahuje jednu veřejnou metodu `Read`, která čte v audio souboru. Jednotlivé vzorky se po rámcích načítají do vyrovnávací paměti po dvaceti milisekundách. Velikost rámce, a tedy i vyrovnávací paměti je spočítána podle vzorkovací frekvence. Pro každý rámec se volá metoda `GetHitTime`, která využívá třídu `HitsDetector` (popsaný dále)

k detekci úderu v rámci a pokud ho detekuje, spočítá čas, ve kterém se rámec vyskytuje. Seznam časů se vrací jako výsledek detekce.

`HitsDetektor` je třída analyzující rámec audiosignálu. Obsahuje veřejnou metodu `HasHit` (na ukázce 4.4), která slouží k detekování, zda se v rámci vyskytuje úder. Třída dále obsahuje několik metod pro analyzování rámce:

- **ComputeFrameEnergy** – spočítá okamžitou energii rámce
- **ComputeAverageLocalEnergy** – spočítá průměrnou energii uloženou v kolekci historyOfEnergy, kam se ukládá historie několika spočítaných energií rámců
- **ComputeSensitivity** – spočítá citlivost
- **ShiftHistoryBuffer** – Posune kolekci historyOfEnergy o jeden prvek doleva a tím umožní uložit poslední spočítanou energii

```
public bool HasHit(float[] samples)
{
    var convolve = Convolution.Convolve(samples, mask);
    var energy = ComputeFrameEnergy(convolve);
    var localEnergy = ComputeAverageLocalEnergy();
    var sensitivity = ComputeSensitivity(samples, localEnergy);
    ShiftHistoryBuffer();
    historyOfEnergy[historyOfEnergy.Length - 1] = energy;

    if (energy > 0.7)
        return energy > localEnergy * sensitivity;
    return false;
}
```

Výpis 4.4: Metoda HasHit detekující úder v rámci

MIDI Parser

Tento modul je tvořen třídou `MidiParser`. Slouží k extrakci not v midi souboru a počítání časů jejich výskytů na základě zadané hodnoty BPM. Tato třída má, stejně jako `WavParser`, dva konstruktory, kde jeden bere v parametru cestu a druhý datový proud a veřejnou metodu `Read`, která bere v parametru hodnotu BPM a vrací seznam datových struktur `NoteDTO`, což je třída popisující notu.

Třída využívá knihovnu *NAudio.Midi* a její třídu `MidiFile`, určenou ke čtení MIDI souborů. Ta umožňuje jednoduše přečíst potřebné MIDI události. V metodě `Read` se nejprve zkontroluje, zda je zadané tempo v požadovaném rozsahu, a pokud ne nebo není zadáno vůbec, nastaví se jako tempo výchozí hodnota unožená v midi souboru. Dále se pomocí metody `GetDrumTrack` ze souboru vyberou pouze události související s bicími nástroji. Metoda `GetDrumNoteOn` vybere události *NoteOn* čili začátky hraní not. A nakonec metoda `GetNoteList` spočítá čas těchto událostí a vytvoří z nich seznam `NoteDTO` objektů. Třída `NoteDTO` slouží jako model k popisu noty. Jsou v ní informace o délce noty, její číslo (u bicích se jedná o konkrétní bubny, ne o tón), název, ale hlavně absolutní pozici v čase. Tento seznam je poté návratovou hodnotou metody.

```
public List<NoteDTO> Read(int bpm = 0)
{
    if (bpm >= 60 && bpm <= 350)
        Tempo = bpm;
    else
        Tempo = (int) TempoEvent.Tempo;

    var drumTrack = GetDrumTrack(midi.Events);
    var drumNoteOn = GetDrumNoteOn(drumTrack.ToList());
    var noteList = GetNoteList(drumNoteOn);
        return noteList;
}
```

Výpis 4.5: Metoda Read pro extrakci not

Porovnávání detekovaných úderů s extrahovanými notami

Modul určený k porovnání a vyhodnocení výstupů MIDI a WAV Parserů je tvořen třídou `Comparer`. Obsahuje metodu `Compare`, která v parametru bere seznam `NoteDTO` objektů a seznam časových značek detekovaných úderů reprezentovaných strukturou `TimeSpan`. Pomocí metody `CreateNoteSegments` se vypočítají informace o tom, v jakých časech začíná a končí okolí noty (pomlky) a tedy, v jakém intervalu se mají (nebo nesmí v případě pomlk) vyskytovat údery. Tyto informace se pak doplní ke každé notě. Z těchto informací se vloží do objektu `NoteSegment`.

Pro každý segment se pak hledají všechny údery nacházející se v daném intervalu, přičemž v každém musí být právě jeden čas úderu anebo žádný, jedná-li se o pomlku. Pokud pro daný segment existuje pouze jediný úder, pak se ještě spočítá, o kolik je tento úder posunutý vůči absolutnímu času noty v segmentu. `NoteSegmet` spolu s rozdílem detekovaného a absolutního času jsou reprezentovány třídou `NoteComapreResult`.

Návratovou hodnotou metody `Compare` je třída `CompareResult`. Ta obsahuje seznam `NoteComapreResult`, tempo, procentuální úspěšnost zahraného cvičení.

4.5 Testování aplikace Drum Learning

Testování je, vzhledem k rozsahu a složitosti softwaru v dnešní době, nezbytnou součástí jeho vývoje. Pomáhá odhalit chyby a ověřit, zda je aplikace vhodně navržená. Výsledný software je nutné testovat neustále a na základě zpětné vazby uživatelů jej upravovat.

Testování této aplikace je rozděleno na dvě části. První se zabývá testováním navržené metody detekce úderů a porovnávání a porovnáváni s midi. Druhou částí jsou pak praktické testy, prováděné na uživatelích.

Testování knihovny pro práci s audiem

V kapitole 3 byl popsán způsob testování navržené metody. Jelikož je logika metody uzavřena v samostatném projektu, je snadné ji testovat samotou automatizovaným způsobem. Součástí řešení jsou dva .NET projekty, které slouží pro testování knihovny *AudioProcessing*. Projekt *AudioProcessing.Tests* pro testování tříd a jejich metod s použitím jednotkové testů *xUnit*. A *AudioProcessing.Console*, což je konzolová aplikace komunikující s knihov-

nou, která byla využívána převážně začátku implementace k rychlému spouštění metody, bez nutnosti používat mobilní aplikaci.

Jednotkové testy (Unit testy) obvykle testují univerzální knihovny. Testují třídy, přesněji jejich metody, jednu po druhé. Předávají jim různé vstupy a zkouší, zda jsou jejich výstupy korektní. *xUnit*[5] je open-source nástroj určený k jednotkovému testování pro .NET. Použití tohoto balíčku v prostředí Visual Studio umožňuje spustit všechny testy a najednou a přehledně procházet jejich výsledky.

Byly napsány jednotkové testy pro testování detekce úderů, tedy třídy `WavParser`. K tomu byla vytvořena sada dat – audionahrávek s údery zahrané na tréninkové pady a vířivé bubny a nahrané na více zařízení. Nahrávky pro testování detekce úderů obsahují jeden, dva nebo žádný úder a pro každou se testuje, zda metoda `Read` třídy `WavParser` vrátí odpovídající počet časových značek, reprezentující detekovaný úder. Na ukázce 4.6 lze vidět metoda testování nahrávky s jedním úderem zahraný na velký pad. Metoda musí být označena atributem *Fact* a k porovnávání výstupu s očekávanou se používají statické metody na třídě `Assert`.

```
[Fact]
public void Read_Drummer_Big_Pad_One_Hit_EqualsTest()
{
    var wavParser = new
        WavParser(Properties.Resources.drummer_big_pad_one_hit);
    var timeSpans = _wavParser.Read();
    Assert.Single(timeSpans);
}
```

Výpis 4.6: Metoda pro testování detekce úderů

Otestování porovnávací metody probíhalo obdobně. Jako testovací data byly vytvořeny soubory midi a pro každý z nich nahrávky v několika variantách:

- Nahrávky vygenerované přímo z midi, tedy nejpřesnější.
- Nahrávky zahrané na tréninkový pad bubeníkem
- Nahrávky nahrané na vířivý buben bubeníkem

Každá tato sada existuje ve variantě zahrané přesně podle MIDI předloze a variantě obsahující předem známý počet chyb. Stejně jako u testování detekce, i zde se kontroluje očekávané hodnocení ke každí testované nahrávce.

Výsledky jednotkových testů

Tabulky 4.1, 4.2 a 4.3 ukazují výsledky jednotkových testů. Jsou zde patrné problémy s nahrávkami vířivého bubnu. V testech detekce úderů selhaly dva testy s nahrávkou dvou úderů na vířivý buben, a to jak v nahrávce pořízené na Xaiomi telefon tak na iPhone. V obou případech byl detekován pouze jeden úder. To se projevilo i testování porovnávací metody, která v případech, kdy je detekován požadovaný počet úderů, tak funguje dobře.

[5] https://xunit.net

Typ testu	Celkem	Selhalo	Úspěch
Jeden úder	7	0	100%
Dva údery	7	2	71,4%

Tabulka 4.1: Výsledky testů podle počtu úderů

Typ testu	Celkem	Selhalo	Úspěch
Snare	7	2	71,4%
Pad	5	0	100%
Side stick	2	0	100%

Tabulka 4.2: Výsledky testů podle typu bubnu

Typ testu	Celkem	Selhalo	Úspěch
Bubeník	7	2	71,4%
Syntetizováno	5	0	100%

Tabulka 4.3: Výsledky testů podle typu typu záznamu

Testovací data se dvěma údery zahrané na vířivý buben, byla doplněna o nahrávky s údery ve rozdílných intervalech. Bylo zjištěno, že pokud se v nahrávce vyskytují dva údery s intervalem kratším než 0,3 s, detektor je vyhodnotí jako jeden úder. To je pravděpodobně způsobeno tím, že zvuk úderu na vířivý buben delší dobu přeznívá a implementovaný detektor je nedokáže správně rozpoznat, pokud se překrývají.

Praktické testy

Praktické testování umožňuje zjistit použitelnost a intuitivnost aplikace v praxi. Testování probíhá na uživatelích a má za účel zjistit, jak dobře jsou schopni ji používat. Cílem je najít a opravit chyby, kvůli kterým se uživatel v aplikaci ztrácí nebo mu brání ke správnému použití, ať už se jedná o chyby funkcionální nebo špatné rozložení prvků na obrazovce. Přiměje vývojáře podívat se na aplikaci z uživatelova úhlu pohledu a ukáže to, co přehlíží.

Byť byl zvolen multiplatformní přístup a výsledná aplikace sdílí převážnou většinu kódu pro platformu Android i iOS, testování probíhalo pouze pro první zvolenou. Je to z důvodu absence iMac zařízení, které je pro nasazení na iPhone nezbytné. V tomto ohledu nebylo zjišťováno, zda aplikace na iOS zařízení funguje.

Testování probíhalo na zařízeních *Xiaomi Redmi 8T* (Android 10), *Xiaomi Redmi Mi A1* (Android 9) *Huawei Mate 10 Lite* (Android 8.0) a *Vivo y20* (Android 11). Při testování na Xiaomi byla zjištěna nefunkčnost nahrávání audia. Tento problém se však na jiných zařízeních neprojevoval. Bylo vyzkoušeno více zařízení Xiaomi a na žádném z nich se nepodařilo pořízenou nahrávku přehrát. Ukázalo se, že je chyba v knihovně *Plugin.AudioRecorder*, která nepřesně zapisuje hlavičku WAV souboru, což na zařízeních Xiaomi působí problém. Tato chyba již byla reportována na *GitHubu*[6]. Knihovna NAudio, použitá při implementaci ke čtení WAV souborů, však zvládne z tohoto poškozeného souboru získat užitečná data. Nakonec se tedy podařilo aplikaci zprovoznit na všech testovaných zařízeních.

[6] https://github.com/NateRickard/Plugin.AudioRecorder/issues/53

Testovací protokol

Pro dobré testování je důležité připravit si otázky a úkoly, které mají uživatelé plnit. Kromě toho je vhodné ponechat uživateli čas na vyzkoušení aplikace a sledovat, jak ji používá. Uživateli bude předáno zařízení se spuštěnou aplikací na hlavní obrazovce, poté bude mít čas na samostatné seznámení se s aplikací. U tohoto kroku bude nahlas komentovat, co dělá. Následuje seznam úkolů, které bude muset samostatně splnit. A na konec bude s uživatelem probíhat diskuse. Seznam úkolů bude následující:

- Pročíst si učební text lekce
- Nahrát cvičení v tempu 60 BPM
- Zobrazit výsledky cvičení
- Vrátit se a zobrazit výsledky jiného cvičení

Testování probíhalo na cílové skupině uživatelů, tedy všichni uživatelé byli bubeníci. Testování se zúčastnilo celkem šest respondentů a probíhalo v hudební zkušebně.

Vyhodnocení praktických testů

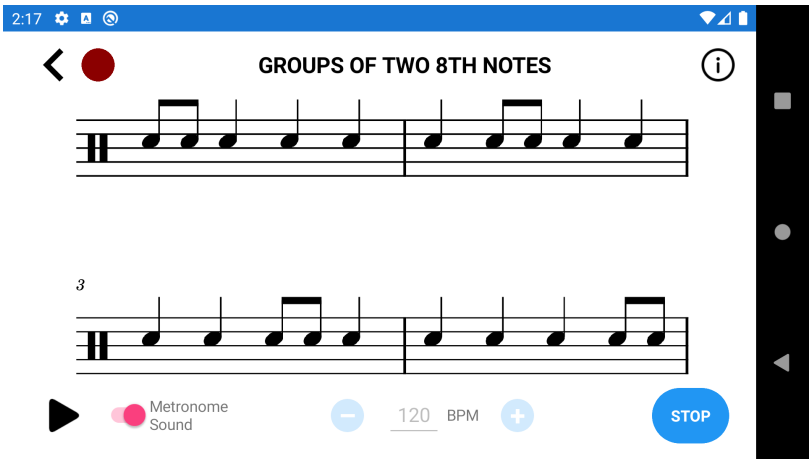
Předpokládalo se a výsledky testů s uživateli prokázaly, že navigace v aplikaci není složitá. Je to dáno malým počtem stránek. Všichni respondenti se bez problému v aplikaci orientovali. Při testování u prvního respondenta byla však v navigaci odhalena první zásadní chyba. Na stránce se cvičením je záměrně skryta navigační lišta. Avšak tato lišta na sobě nese tlačítko pro návrat na předchozí obrazovku. Pro návrat lze na zařízení Android použít systémové tlačítko zpět přímo na telefonu. Na testovaném zařízení byla tato systémová tlačítka skryta, což uživatele zmátlo. Nedostatek byl ihned napraven přidáním vlastního tlačítka pro návrat umístěného na standardním místě vlevo nahoře.

Další chyba byla odhalena při nastavování tempa a spuštění hraní, kdy se jeden z uživatelů pokusil změnit tempo během nahrávání. Ovládací prvky zůstaly aktivní. Na chod aplikace to nemělo žádný vliv, ale uživatele to zmátlo.

Při spuštěném nahrávání se zároveň zapne i metronom, který každý svůj tik signalizuje probliknutím červeného kruhu a přehráním zvuku kliknutí. Tato funkce je užitečná pro pomoc udržení rytmu. Avšak se ukázalo, že pokud je hlasitost zařízení (kliknutí metronomu) vysoká, je tento zvuk rozpoznán jako úder. Do aplikace byla přidána možnost tento zvuk vypnout a ponechat tak jen vizuální signalizaci metronomu. Akustická signalizace lze použít se sluchátky.

Nastavení tempa a spuštění nahrávání jinak respondenti zvládli bez komplikací. Stejně tak zobrazení výsledků. Uživatelům vykreslený výsledek v notách přišel pochopitelný.

Výše uvedené poznatky byly do aplikace zaimplementovány. Na obrázku 4.4 je finální obrazovka se cvičením. Je zde vidět přidané tlačítko zpět vlevo nahoře. Hned vedle se nachází vizuální signalizace tiku metronomu. V dolní části vedle tlačítka *Play* je přepínač přehrávání zvuku metronomu. A nastavení BPM při nahrávání je deaktivováno.



Obrázek 4.4: Finální podoba obrazovky cvičení

Kapitola 5

Závěr

Cílem této práce bylo navrhnout, implementovat a otestovat mobilní aplikaci, která bude sloužit jako podpora při výuce hry na bicí nástroje. Hlavní myšlenkou aplikace bylo poskytnout uživateli zpětnou vazbu o jeho hraní.

Úvod práce se věnuje představením základní terminologie týkající se hry na bicí nástroje. Bylo potřeba najít a prozkoumat existující aplikace, seznámit se s postupy zpracování audio signálů, s rozhraním MIDI a formátem souborů SMF. Na základě těchto poznatků navrhnout metodu detekce úderů bubnu a jeho porovnání s MIDI souborem. Dále probíhalo seznámení se s mobilními platformami a technologiemi pro vývoj mobilních aplikací, kde bylo zvoleno multiplatformní řešení v technologii Xamarin.Forms. Nejrozsáhlejší částí bylo studium technologie Xamarin a .NET, implementace aplikace a testování funkcionality. V kapitole zabývající se implementací je představena architektura celé aplikace, popis jejích nejdůležitějších částí a vizuální podoba aplikace.

Nedílnou součástí bylo testování. Byly napsány automatické testy pro ověření funkčnosti detekce úderů a po celou dobu vývoje aplikace byla metoda na základě výsledků upravována. Při detekci úderů do tréninkového padu bylo dosaženo velmi uspokojivých výsledků. U úderů na vířivý buben byl odhalen nedostatek v rozpoznávání úderů, opakujících se rychle po sobě. Mobilní aplikace byla testována několika bubeníky, jejichž poznatky byly zaimplementovány.

Podařilo se navrhnout jednotlivé části a implementovat prototyp výukové mobilní aplikace, která poslouží jako základ pro další vývoj. Ačkoliv se volba multiplatformního řešení původně jevila jako vhodná, ukázalo se, že technologie Xamarin.Forms se pro tento typ aplikací příliš nehodí.

Do budoucna by bylo dobré zvážit použití jiné technologie, nebo zdali se takováto aplikace nevyplatí vyvíjet spíše nativně pro konkrétní platformou s plnou podporou aplikačního rozhraní. Jako další prací na aplikaci je určitě vylepšování metody rozpoznávání úderů, které by se mohlo rozšířit na detekci více různých bubnů. Dále by to mohla být implementace lepšího zobrazování not nebo se třeba při výuce zaměřit na gamifikaci.

Literatura

[1] *Drums schematic* [online]. Wikipedia, říjen 2013 [cit. 2021-05-09]. Dostupné z: https://en.wikipedia.org/wiki/File:Drums_schematic.svg.

[2] *Drums: real drum set music games to play and learn* [online]. MWM - Free music and creative apps for Android, květen 2021 [cit. 2021-05-09]. Dostupné z: https://play.google.com/store/apps/details?id=com.mwm.drum&hl=en&gl=US.

[3] Bouffioux, L. *Xamarin SKIASharp: Guide to MVVM* [online]. Code Project, září 2019 [cit. 2021-05-03]. Dostupné z: https://www.codeproject.com/Articles/5247780/Xamarin-SKIASharp-Guide-to-MVVM.

[4] Dajbych, V. *Mvvm: model-view-viewmodel* [online]. dotNETPortal.cz, duben 2009 [cit. 2021-05-03]. Dostupné z: https://www.dotnetportal.cz/clanek/4994/MVVM-Model-View-ViewModel.

[5] Drum Center of Portsmouth. *Parts of a Drum Set—The Structure & What You Should Know* [online]. Drum Center of Portsmouth, listopad 2020 [cit. 2021-05-03]. Dostupné z: https://www.drumcenternh.com/news/drum-set-parts/.

[6] Gamedev.net. *Beat detection Algorithm* [online]. Gamedev.com, © 1999-2011 [cit. 2021-05-03]. Dostupné z: http://archive.gamedev.net/archive/reference/programming/features/beatdetection/index.html.

[7] Loudová, I. *Moderní notace a její interpretace*. Praha: Akademie múzických umění, 1998. ISBN 80-85883-31-7.

[8] Metev, D. *39+ Smartphone Statistics You Should Know in 2020* [online]. review42.com, únor 2021 [cit. 2021-05-03]. Dostupné z: https://review42.com/resources/smartphone-statistics/.

[9] Microsoft. *Handle and raising events* [online]. Microsoft, březen 2017 [cit. 2021-05-03]. Dostupné z: https://docs.microsoft.com/en-us/xamarin/xamarin-forms/app-fundamentals/data-binding/binding-mode.

[10] Microsoft. *SkiaSharp Graphics in Xamarin.Forms* [online]. Microsoft, září 2017 [cit. 2021-05-03]. Dostupné z: https://docs.microsoft.com/en-us/xamarin/xamarin-forms/user-interface/graphics/skiasharp/.

[11] Microsoft. *Xamarin.Forms XAML Basics* [online]. Microsoft, říjen 2017 [cit. 2021-05-03]. Dostupné z: https://docs.microsoft.com/en-us/xamarin/xamarin-forms/xaml/xaml-basics/.

[12] Microsoft. *Xamarin.Forms Binding Mode* [online]. Microsoft, leden 2018 [cit. 2021-05-03]. Dostupné z: https://docs.microsoft.com/en-us/xamarin/xamarin-forms/app-fundamentals/data-binding/binding-mode.

[13] Microsoft. *What is Xamarin?* [online]. Microsoft, květen 2020 [cit. 2021-05-03]. Dostupné z: https://docs.microsoft.com/en-us/xamarin/get-started/what-is-xamarin.

[14] Microsoft. *XAML overview (WPF .NET)* [online]. Microsoft, duben 2021 [cit. 2021-05-03]. Dostupné z: https://docs.microsoft.com/en-us/dotnet/desktop/wpf/xaml/?view=netdesktop-5.0.

[15] MLČOCH, M. *Zpracování akustických signálů v bezpečnostních technologiích [online].* 2019. [cit. 2021-05-03]. Diplomová práce. Univerzita Tomáše Bati ve Zlíně, Fakulta aplikované informatikyZlín. Vedoucí práce Mgr. Milan Adámek, P. doc. Dostupné z: https://theses.cz/id/hazrdn/.

[16] Moore, F. R. An Introduction to the Mathematics of Digital Signal Processing: Part II: Sampling, Transforms, and Digital Filtering. *Computer Music Journal* [online]. The MIT Press. Září 1978, sv. 2. Dostupné z: https://www.jstor.org/stable/3680221?seq=1.

[17] Richardson, G. *Drums Anatomy – Parts of a Drum Set Explained* [online]. Zing Instruments, duben 2021 [cit. 2021-05-03]. Dostupné z: https://zinginstruments.com/parts-of-a-drum-set/.

[18] Tišňovský, P. *General MIDI a formát souborů SMF* [online]. Root.cz, březen 2009 [cit. 2021-05-03]. Dostupné z: https://www.root.cz/clanky/general-midi-a-format-souboru-smf/.

[19] Tišňovský, P. *Rozhraní MIDI na osobních počítačích* [online]. Root.cz, únor 2009 [cit. 2021-05-03]. Dostupné z: https://www.root.cz/clanky/rozhrani-midi-na-osobnich-pocitacich/.

[20] Černocký, J. *Úvod do signálů a systémů* [online]. FIT VUT v Brně, 2020. Dostupné z: https://www.fit.vutbr.cz/study/courses/ISS/public/opora/iss.pdf.